UNIVERSITY OF JYVÄSKYLÄ

1650

COMPILATION OF LANGUAGE CORPORA: COMPUTER RELATED ISSUES AND ANNOTATION

A Pro Gradu Thesis

by

Eeva Marin

Department of English
1999

CONTENTS

HUMANISTINEN TIEDEKUNTA
ENGLANNIN KIELEN LAITOS

Eeva Marin
COMPILATION OF LANGUAGE CORPORA:
COMPUTER RELATED ISSUES AND ANNOTATION

Pro Gradu –tutkielma
Englantilainen filologia
Kesäkuu 1999 195 sivua + 2 liitettä

Tämän tutkimuksen tarkoituksena on toimia oppaana kielikorpusten koostamisessa. Työ liittyy osittain FECCS (Finnish-English Contrastive Corpus Studies) –korpukseen, joka on koottu Jyväskylän yliopiston Englannin kielen laitoksella. Lukijaksi on ajateltu lähinnä korpuksista kiinnostunutta kielitieteilijää, jolla on tietokoneiden peruskäyttötaito mutta joka ei ole niiden asiantuntija.

Tutkimus keskittyy korpusten koostamiseen. Painopiste on nimenomaan koostamisen teknisessä puolessa: tietokoneisiin liittyvissä kysymyksissä ja annotaatiossa.

Tutkimuksessa on alkuintron lisäksi kaksi pääosaa. Introa seuraava työn toinen osa liittyy tietokoneita koskeviin kysymyksiin. Siihen sisältyy mm. yleisiä ohjeita tietokoneiden ja käyttöjärjestelmien käytöstä sekä tietoa merkistöistä, tiedon tallentamisesta ja korpusohjelmien hankkimisesta. Kolmannessa osassa kuvataan eri tyyppisiä annotaatioita. Siinä käydään läpi mitä erilaisia annotaatioita on olemassa, mihin niitä käytetään ja miten niitä yleisesti ottaen lisätään korpukseen. Kumpikin näistä suuremmista osista myös kuvaa minkälaisia ongelmia korpuksen koostamisessa voi tulla. Tutkimuksessa kuvataan myös kokemuksia FECCS-korpuksen koostamisessa.

Asioita käsitellään kohtalaisen yleisellä tasolla. Tarkoituksena ei ole keskittyä mihinkään tietyntyyppisiin tai –kokoisiin korpuksiin, joskin ehkä monin kohdin oletuksena on kohtalaisen pieni tai keskikokoinen korpus. Yleisellä tasolla pysytään myös siksi, ettei moniin tietokoneita koskeviin ongelmiin voi antaa ehdottomia vastauksia laitteiston ja ohjelmistojen nopean kehityksen takia. Siten tässä tutkimuksessa pyritään lähinnä antamaan yleisiä ohjeita, jotka sitten toisaalta säilyttävät pätevyytensä pitemmän ajan.

Asiasanat: corpus, corpus compilation, annotation, electronic texts, bilingual corpora, TEI

1 INTRODUCTION

The present paper is about the compilation of computer corpora. A corpus is a large collection of text in computerized form that can be searched and otherwise manipulated by the computer. Corpora often include millions, even hudreds of millions of words that have been collected from various sources, such as books, newspapers and magazines. They can be used for many different kinds of linguistic research and more practical applications for them can be found in the fields of translating and teaching, among other things.

The idea for such a paper as this stems from my working within a corpus project. A corpus called FECCS (Finnish-English Contrastive Corpus Studies) has been compiled at the Department of English at the University of Jyväskylä. The finished FECCS corpus will include extracts of texts in both English and Finnish, and their Finnish and English translations. The final size of the corpus will be about 2 million words.

The FECCS corpus has been compiled together with the Universities of Oslo and Bergen in Norway, and the University of Lund in Sweden. They have, respectively, English-Norwegian and English-Swedish corpora. The original English texts are, to a great extent, the same extracts in all three projects. There is also a smaller corpus at the Department of English in Jyväskylä that consists of newspaper articles from tabloid newspapers such as Daily Mirror, The Sun, Iltalehti and Ilta-Sanomat.

It was noted during the compilation of the corpus that there does not seem to be much written on the subject of compiling a corpus. The existing studies either concentrate on the selection of texts rather than the technical side of the compilation process, or, conversely, tend to be highly technical and closely connected to a specific project. There did not seem to be collected general guidelines that would apply to all types of corpora and at the same time be suitable for someone who does not know much about the field beforehand. There were bits of information available here and there, but never were those bits collected under one heading. From this basis came the idea of writing the present paper, which tries to collect relevant points on corpora compilation into a single collection of information, advice and experiences.

Basically, the present paper aims to give a general idea of the process of compiling a corpus. The purpose is to provide a general introduction to corpus compilation to someone who does not yet know much about the field. It is expected that the readers are familiar with the basics of linguistics; also, it is expected that the readers are not total strangers with computers. The present paper does not, however, expect expert knowledge about computers or programming. In fact, for computer experts, some of the chapters may seem rather simplified and tedious. Rather, it is expected that the reader is familiar with some of the basic applications of computers, such as file management and word processing. It is also expected that the reader understands the basics of how computer software work, so that terms such as "cutting and pasting" text are familiar. Mainly, the present paper is aimed at linguists who are interested in taking advantage of computers in linguistics, especially in the form of corpora. It is also mostly expected that it is the linguists themselves who need to take care of practical computer issues, and that they use personal computers rather than large mainframe systems.

The present paper aims to describe *what* can be done to corpora rather that *how* it can be done, exactly. For example, chapter 3.5 introduces what part-of-speech annotation is, explains in general terms how it is done, and what kinds of problems can be expected with it. It does not, however, give much practical advice about how to do it. There are mainly two reasons for this.

First of all, there rarely is only one "absolutely correct" way to do something. Part-of-speech annotation, for example, can be applied in several different ways: it would not serve the purposes of the present paper to introduce one of them here as being above all the others. The annotation schemes, in their entirety, are also quite extensive: it might take hundreds of pages to explain even one of them, in some cases. If there is a need to get to know a specific annotation scheme, there are published manuals for many schemes that tell everything about them.

Secondly, the development of computers is rather rapid nowadays. Should the present paper represent a particular procedure in a very detailed manner, its usefulness would not last for very long. It is much more useful to

explain some of the basic principles; even though the details change, many of the basic guidelines for doing something are likely to last much longer.

It must be noted here, though, that some examples of the present paper are nevertheless tied to the time of writing. For instance, the chapter about operating systems may seem rather dated quite soon, since the changes in the field are fast. It, too, should be read so that attention is paid to some of the main ideas rather than the details. The details (such as the descriptions of specific operating systems) may become outdated soon; some of the more general ideas (such as what kinds of qualities a researcher in the humanities should expect from an operating system) are bound to last much longer.

It must also be pointed out that much of the information in the present paper is on a rather theoretical level. Especially Part Three of the present paper is mostly based on what I have read about annotating corpora; I have not personally had the chance to use the annotations that are discussed, with the exception of alignment that was done to the FECCS corpus. Those who have actually worked with annotated corpora could undoubtedly offer a somewhat different view to the matters that are discussed here. As it is, the information on the present paper is based on such sources as were available.

The first part of the present paper serves as an introduction to corpus linguistics. It explains what corpora are, what they can be used for, and presents some key vocabulary in the compilation and use of corpora. It is intended as a general background to someone who is not yet familiar with corpus linguistics at all.

The second part of the present paper is about relevant computer-related issues. Since corpora are nowadays kept in computers, there are many computer-related questions that need to be addressed when compiling a corpus. These include, for example, suitable hardware and software, and questions relating to storing the information. The second part also addresses the question of character sets, which sometimes cause problems with text files. Part Two of the present paper, especially, is meant for readers who have a limited knowledge of computers in advance.

Part Three of the present paper examines the field of annotation. Annotation means the addition of extratextual information to the corpus. For

example, each word in the corpus could be provided with a code that tells which word-class it belongs to, or each sentence in a corpus could be syntactically analyzed. The beginning of Part Three, first of all, explains how to get the texts into the computer: the subsequent chapters, then, tell what kind of information can be added to the texts once they are in computerized form.

There are also many things that the present paper does *not* deal with. The most notable of these is the criteria for the selection of the contents of the corpus. As will be pointed out, a corpus is a *principled* collection of texts, so that there is some kind of an idea behind their selection. For example, the texts could be selected from 19th century literature, or from certain type of newspapers that were published, say, in 1990. In addition to the fact that a corpus should be principled, there is the question whether the corpus is *representative* of the field it is supposed to present. For instance, it may be desired that the selected texts from 19th century literature are representative so that the research results from the corpus can be generalized to apply to any kind of 19th century literature. There is also the question of what kind of *samples* are selected for the corpus: are a few complete novels included, or extracts from several novels. These are the kinds of questions that the present paper will *not* address any further. In the present paper, it is expected that the selection of texts has been made. There seems to be plenty of research done already in the field of corpus design. Those who wish to know about the selection of texts for the corpus are referred to relevant research reports. Short descriptions of such matters can be found in Biber et al. (1998:246-253) and in Barnbrook (1996:24-25). Also Sinclair (1991:15-21) discusses corpus design. The first part of Leitner (ed.) (1992) deals with corpus design and text encoding, and includes articles on, for example, corpus sampling. An example of the text selections of an individual project can be found, for example, in Biber et al. (1993) (concerning a representative corpus of historical English registers).

Another field that is not much addressed here is the actual usefulness of specific types of operations that can be done to or with corpora. For example, the usefulness of different types of annotations should not be taken for granted; what kinds of annotations are useful depends on the type of

research that is going to be done with the corpus. There are, indeed, many questions and problems in the use of corpora that could be addressed. It could be considered, for example, is a specific corpus appropriate for a specific type of research, does the corpus dictate what kinds of research can be done with it, how much do the programs that are used limit the kind of research that can be done, and are the research results poor because there was no search option for the thing that was really wanted. Maybe the results are not at all what was wanted, or maybe the computer retrieved too much, or too little. It may be that a corpus is actually not necessary at all for a given research question; rather, the matter could be examined much more efficiently by other methods. These kinds of questions are very much related to specific research projects. Even on a general level, however, they will not be addressed in the present paper. The chapters on annotation do give some examples of the uses of the types of annotations introduced, but they do not go very far in criticizing the usefulness or theoretical basis of the annotation schemes. Such issues, though interesting, are out of the bounds of the present paper.

The following chapters of the first part of the present paper introduce corpus linguistics in general. Chapter 1.1 gives general background information about computer corpora and their different types. Chapter 1.2 is about terminology: it explains some central terms in corpus linguistics and gives examples of them. Chapter 1.3 introduces a few programs that can be used in corpus research, to give a further idea of how corpus software works. Chapter 1.4, in turn, explains some of the uses of corpora in general. Note that the uses for annotated corpora are considered in connection of the relevant types of annotations in Part Three of the present paper. Chapter 1.5 briefly deals with the issue of copyrights.

1.1 Background on corpora

A corpus is a collection of texts or any language material that has been systematically collected for a specific purpose in mind, most often for research. A corpus usually includes millions of words of text in machine-readable form that can be accessed through suitable computer programs. The computer can search the texts for a particular word or phrase, and the findings can be printed out or saved on disk. A search is usually made for one word, and the computer prints every sentence in which the word was found.

Glossary of Corpus Linguistics (at the Cobuild WWW site) defines a corpus as "a collection of naturally-occuring language text, chosen to characterize a state or variety of a language [...]". This definition actually includes the same important point that Leech and Fligelstone (1992:116) also mention: corpora are rarely collections of miscellaneous texts; rather, the types of texts in them have usually been carefully selected and classified so that the corpus is representative of some language or text type. Biber et al. (1998:12) put all this in an efficiently succinct form: a corpus is a *principled* collection of *natural* texts. For example, a corpus may include newspaper articles from certain newspapers from a certain period of time.

There are some well-known corpora in the world. The Brown corpus and The LOB (Lancaster-Oslo/Bergen) corpus are both from the early sixties, and both comprise about one million words. Although they are quite old, they are still widely used; because their contents is well-known by now, they can be used, for example, to test new programs and other innovations. They are both fairly small when compared to newer corpora. The British National Corpus contains about 100 million words in all; The Bank of English contains several hundreds of millions. There is also a well-known corpus in Finland, the Helsinki corpus. It includes, for example, Old and Middle English texts.

A corpus may include both written and spoken language. In case of spoken language, speech must be first transcribed into written form in order to make it possible to examine it on a computer. Sinclair (1991:15-16) notes that although many researchers believe that spoken language would tell more about the fundamentals of language that written texts, most corpora do not include spoken language because of the various problems in compiling such material.

There are still no methods for automatic transcription of speech and therefore the compilation of such a corpus is very time-consuming. There are, however, some well-known corpora that include spoken language. These include the London-Lund corpus (500,000 words of spoken language) and the British National Corpus (4,000,000 words of spoken language) (Biber et al. 1998:13).

Most of the well-known corpora in the world contain English texts, but there are also corpora in other languages. There are also attempts to make corpora more standardized in terms of how annotation is applied and what kinds of programs can be used. With more standard practices, it will be possible to combine corpora from several different countries and in many languages to form larger corpora that could be used for multilingual research. Chapter 3.3 of the present paper will consider the matter of standards in more detail.

Although the term "corpus" nowadays usually refers to something that is in a computerized form, the concept of corpus linguistics has been around for several decades. It is only the recent fast development of computers that has made it possible to compile such huge corpora that are around now. Earlier, a corpus used to mean a collection of texts in printed form. Several such books, composed of hundreds and hundreds of pages of collected texts, can still be found in libraries. Doing research on such a corpus meant that the researcher had to manually search for every single instance of whatever he was looking for.

The development of both computer hardware and software has had a huge effect on both the amount of material available and what can be done with corpora. Within the last fifteen or twenty years there has been such an incredible increase in computing power that it is hard to describe. Sinclair (1991:1) gives an impressive account of the development of corpus linguistics:

> "Thirty years ago when this research started it was considered impossible to process texts of several million words in length. Twenty years ago it was considered marginally possible but lunatic. Ten years ago it was considered quite possible but still lunatic. Today it is very popular."

Although the uses of corpora will be explained in more detail later in this paper, it might be noted here that corpora can be used in several kinds of linguistic research, as well as to support language learning and translation. For

example, many students who study English at a university are familiar with *The Collins Cobuild English Language Dictionary*. This dictionary is the result of the Cobuild project by the University of Birmingham and HarperCollins Publishers (Sinclair 1991:1-3, *Glossary of Corpus Linguistics*). The numerous examples in the dictionary come from their huge corpus of both written and spoken language. Nowadays, the Cobuild database, called Bank of English, contains over 320 million words (*The Bank of English – Questions and Answers*).

The following sections describe the different types of corpora there are. It is useful to keep the classification in mind. Although there are many characteristics that all corpora have in common, in many respects different types of corpora are suitable for rather different purposes, and the bases for their compilation are different.

1.1.1 Different types of corpora

There are more than one kind of corpora in existence. The different types of corpora can be classified in several ways. The classifications tend to depend on the point of view of the writer and the kind of research being done. The classification presented here is one that seems most relevant for the purposes of the present study. The main division is into monolingual and bilingual corpora.

1.1.1.1 Monolingual corpora

Monolingual corpora are, as the name implies, corpora that consist of samples from one language only. Most corpora in the world are monolingual. The Cobuild corpus is a typical example of this kind of corpus. Although it spans several text types and both written and spoken language, it consist of English language texts only. The Cobuild corpus is what can be called a **general corpus** (e.g. Sinclair 1991:17, Mauranen 1997). It does not specilize in any certain type of text but tries to give a very broad view to the language, including material from newspapers, magazines, fiction and non-fiction books, brochures, leaflets, reports, letters, and spoken language such as transcriptions

of conversation, radio broadcasts, meetings, interviews and discussions (*The Bank of English – Questions and Answers*).

The other type of monolingual corpus is that of a **specialized corpus**. Such a corpus has texts from a specified field or era. For example, the Helsinki Corpus includes a collection of Old and Middle English texts and transcripts of spoken British rural dialects from the 1970's (*Manual to the Diachronic part of the Helsinki Corpus of English Texts)*. There are also corpora that concentrate on the language of a certain technical or other field, for example corpora of legal language.

Yet another type of monolingual corpus is that of a **monitor corpus**. A monitor corpus is a corpus that constantly goes through new stretches of text and searches for new words and expressions. Sinclair (1991:25) suggests that a monitor corpus should always have available an up-to-date selection of current English; when the corpus gets too large, part of the information can be discarded. A monitor corpus is constantly changing.

1.1.1.2 Bilingual and multilingual corpora

A bilingual corpus is a corpus that has texts in two different languages. A multilingual corpus, respectively, has more than two languages in it. The same classifications apply to both bilingual and multilingual corpora, and henceworth the present paper will, for the sake of brevity, mostly refer to them as bilingual corpora. There are two main kinds of bilingual corpora: translation corpora and comparable corpora.

A **translation corpus** includes original texts and their translations. They are sometimes also called **parallel corpora**, although the term can refer to other types of corpora, too. As Peters and Picchi (undated) define it, the texts in a translation corpus are translationally equivalent. As was mentioned earlier, there is a corpus of English and Finnish texts, the FECCS (Finnish-English Contrastive Corpus Studies) corpus, at the Department of English at the University of Jyväskylä. More than half of the texts are English originals with their Finnish translations, and the rest are Finnish original texts and their English translations. Similar corpora have been collected at the University of Oslo in Norway (ENPC, English-Norwegian Parallel Corpus) and at the

University of Lund in Sweden, with English-Norwegian and English-Swedish texts, respectively. These corpora consist mainly of prose literature and popular fiction.

By using the computer programs made for this purpose, a researcher can find any sentence from a text and its translation in the other language. Translation corpora have often been processed beforehand so that corresponding sentences can be found easily and fast. A researcher can search for any word or phrase in either language, and the computer gives a list of the sentences where the word occurs and their corresponding translations.

The other type of a bilingual corpus is that of a **comparable corpus**. Peters and Picchi (undated) define a comparable corpus as consisting of "sets of texts [...] that concern a given domain and can be contrasted because of their common features". The texts in a comparable corpus share common features such as period of time, topic and register. They are not, however, translationally equivalent and cannot be examined in the same way as a translation corpus. There is also a small comparable Finnish-English corpus at the Department of English in Jyväskylä. It consists of articles from such Finnish tabloid newspapers as *Iltalehti* and *Ilta-Sanomat*, and British papers such as *Daily Mirror* and *The Sun*. The articles were all selected from papers that were published within a period of few months, and similar types of news were chosen (e.g. editorial, sports news).

Like monolingual corpora, bilingual and multilingual corpora can be, in fact, specialized in a certain field. In Aarhus School of Business in Denmark, there is a Danish-English-French corpus within the domain of contract law and a Danish-German-Spanish corpus of genetic engineering (Lauridsen 1996:67).

This chapter has given general background on computer corpora and introduced different types of corpora that are available. Before the present paper moves on to explain more about the uses of corpora, there are some important terms that must be explained. They will be introduced in the following chapter, which will deal with the terminology of both the compilation and use of corpora.

1.2 Terms and concepts

This chapter introduces some key terms of corpus linguistics. Although the present paper deals with the compilation of corpora, there are also some basic terms relating to the use of corpora that need to be introduced. The following section will explain, first, some relevant concepts of compiling corpora, and secondly, terms in using corpora.

1.2.1 Terminology – compiling a corpus

Terms that often come up with corpora are **annotation**, **annotating**, and **annotated corpora**. Almost all of Part Three is dedicated to the matter of annotation, but a few words about it are in place here to make some of the related vocabulary familiar. Annotation means, basically, that some extratextual information is added to a corpus. Annotation is often also referred to as **tagging**, or **markup**. Sometimes a slight difference is made between these terms. The difference between them seems to be that annotation is a more general term, and it is often also used to refer to the process rather than the outcome. Tagging and markup tend to mean either the encoding itself (the codes themselves), or more specific type of encoding, namely the kinds of tags that are attached to each word, sentence, or other smaller units (vs. annotation that describes features of a complete text). In the present paper, these terms will be used interchangably.

In practice, annotation means that special encoding of some kind is added to the text. It may also mean that certain general features of the text, such as bibliographical information, is added to the beginning. The codes that are used are often referred to as **tags**. A tag is a code that usually looks much like something that might be used in programming, which makes the principles of their use easy to comprehend for people who are familiar with computer languages. Tags can be attached to certain words, phrases, sentences, or other strecthes of text. For example, a tag can tell that a word is a proper noun, or a pair of tags can tell where a paragraph starts and ends. Tags can also tell of material that has been omitted from the text, such as pictures.

Annotation can be added either manually or automatically. Manual annotation means that human annotators, with the help of a word processor or another type of editor, put each tag in place. Automatic annotation means that the computer can carry out the whole task more or less by itself. In practice, the annotation process is often a mix of the two. Very often, the computer first assigns tags, and then human annotators check the result and correct all the errors.

Theoretically, almost any characteristics and details of a text could be annotated. In practice, however, some features are easier to annotate than others. For example, information about the grammatical categories of words can be added relatively easily; information about the meanings of phrases or sentences are much harder to annotate. The problem is not only that such markup is very hard to do automatically, since computers do not generally understand the text they process, but also that even for human annotators, it is sometimes impossible to give unambiguous meanings to units of text.

There are many reasons why annotation is added to a corpus, and they are dealt with in more detail in Part Three in the present paper. In general, annotation has consequences on how the texts can be manipulated in corpus programs, what kind of research can be done with them, and how, if needed, the texts can be further processed in order to make a working corpus. Therefore, making a decision about what kinds of annotations are added to a corpus can prove to be rather important for the researchers. The following is an extract from a tagged text from the FECCS corpus:

Example 1: Tagged text

```
<div1 type=part id=PM1.1>
<head>JANUARY</head>
<pb n=1><p><s>The year began with lunch.</s></p>
<p><s>We have always found that New Year's Eve, with its
eleventh-hour excesses and doomed resolutions, is a dismal
occasion for all the forced jollity and midnight toasts and
kisses.</s> <s>And so, when we heard that over in the
village of Lacoste, a few miles away, the proprietor of Le
Simiane was offering a six-course lunch with pink champagne
to his amiable clientele, it seemed like a much more
cheerful way to start the next twelve months.</s></p>
```

(Peter Mayle: *A Year in Provence* (FECCS))

In this particular case, for the FECCS corpus, sentence and paragraph boundaries have been recognized. The <p>-tags single out paragraphs and the <s>-tags surround sentences. There is also the <head>-tag marking the chapter heading, and a <pb>-tag for a page break. The -tag in the beginning defines that this division of the text is a part and its identity number is PM1.1. These tags have been inserted in order to tell the computer where one sentence starts and ends, where one paragraph starts and ends, and where the chapters change. Although for a human reader it might seem obvious where a new sentence starts, for the computer it is not. For example, a full stop and a capital letter do not always start a new sentence, e.g. with names like *C.G.E. Mannerheim* and abbreviations like *esim. Helsinki*. The tags unambiguously tell where sentences and paragraphs start and stop and make the processing of a text easier.

The tags, in this case, are based on SGML (Standard Generalized Markup Language), which is used in structured documents. SGML will be further dealt with in chapter 3.4. The tags also resemble HTML (Hypertext Markup Language) tags, which are used to define documents in the World Wide Web. Most of the tagging in FECCS could be done automatically, although some tags must be inserted manually. All automatically inserted tags must be proofread afterwards, because the computer makes errors with them.

In addition to the tags in the previous example, there are many other tags in the FECCS corpus. They include, for instance, special tags for italic and bold face, tags for pictures and tables (the actual pictures and tables have to be omitted and the tags are inserted instead) and tags for foreign language expressions. The point of SGML tagging here is to preserve the formatting of a text, no matter what kind of program it is processed in.

Another important term that also concerns the FECCS corpus is **alignment**. Alignment is relevant in case of translation corpora. It means that the corresponding sentences in two texts (e.g. an English text and its Finnish translation) are matched so that they can be easily found together. The following examples are from the same text as the previous example of a tagged text (Example 1). Here the texts, an English original and, below it, its Finnish translation, have been aligned.

Example 2: Aligned texts

```
<div1 type=part id=PM1.1>
<head id=PM1.1.h1 corresp=PM1T.1.h1>JANUARY</head>
<pb n=1>
<p id=PM1.1.p1>
<s id=PM1.1.s1 corresp=PM1T.1.s1>The year began with
lunch.</s></p>
<p id=PM1.1.p2>
<s id=PM1.1.s2 corresp=PM1T.1.s2>We have always found that
New Year's Eve, with its eleventh-hour excesses and doomed
resolutions, is a dismal occasion for all the forced jollity
and midnight toasts and kisses.</s>
<s id=PM1.1.s3 corresp=PM1T.1.s3>And so, when we heard that
over in the village of Lacoste, a few miles away, the
proprietor of Le Simiane was offering a six-course lunch
with pink champagne to his amiable clientele, it seemed like
a much more cheerful way to start the next twelve
months.</s></p>

<div1 type=part id=PM1T.1>
<head id=PM1T.1.h1 corresp=PM1.1.h1>TAMMIKUU</head>
<p id=PM1T.1.p1>
<s id=PM1T.1.s1 corresp=PM1.1.s1>Vuosi alkoi lounaan
merkeissä.</s></p>
<p id=PM1T.1.p2>
<s id=PM1T.1.s2 corresp=PM1.1.s2>  Meistä uudenvuodenaatto
yhdennentoista hetken hurjasteluineen ja kohtalokkaine
päätöksineen on aina ollut masentava tilaisuus; päkistettyä
riehakkuutta, keskiyön maljoja ja suudelmia.</s>
<s id=PM1T.1.s3 corresp=PM1.1.s3>Ja kun sitten kuulimme,
että Le Simianen isäntä Lacosten kylässä muutaman mailin
päässä tarjosi rakastettavalle asiakaskunnalleen kuuden
ruokalajin lounaan vaaleanpunaisen samppanjan kera, meistä
tuntui että se olisi paljon hilpeämpi tapa aloittaa seuraava
kahdentoista kuukauden jakso.</s></p>

                    (Peter Mayle: A Year in Provence (FECCS))
```

Each sentence now has its own unique identity number and information about the identity number of the matching sentence. Now the program that is used to browse the corpus can find a sentence in either language and its translation in the other language easily.

There are programs that automatically do the alignment. They may do it by the aid of a wordlist, which is like a concise dictionary where the program can find the words it sees and compare them to their translations. Alignment programs may also work on many other basis, often utilising the sentence lengths and probabilistic information about the likelihood of certain matches. Alignment will be explained in more detail in chapter 3.11.

The following section, in turn, introduces some terms relevant in the use of corpora. Although the present paper does not specifically address the complications that may occur in the use of corpora, occasional references to them are possible. In any case, the terms in the following section are rather central to the field of corpus linguistics, and are therefore introduced here.

1.2.2 Terminology – using a corpus

There are some important terms that one should be familiar with in order to be able to understand the use of corpora. These include **concordance**, **collocation**, and **lemma**. Each of them will be explained here in turn.

1.2.2.1 Concordance

Sinclair (1991:32) defines **concordance** as "a collection of the occurences of a word-form, each in its own textual environment". The following example is part of the concordance of the word *blue* from the English original texts of the FECCS corpus:

Example 3: Concordance, search word *blue* (no sorting)

```
 1  hair dyed straw-blonde or baby-blue, or, even more startling aga
 2   Toronto the Good, Toronto the Blue, where you could n't get win
 3  old biddy. I pull on my powder-blue sweatsuit, my disguise as a
 4  up the caterpillars, which are blue-striped, and velvety and coo
 5 . She blew it up for me. It was blue, translucent, round, like a
 6 hristmas tree lights, yellow or blue or green. These are called "
 7  wearing grey slacks and a dark-blue plaid shirt, packing our foo
 8  on. A soft June morning with a blue sky and a gentle breeze. Six
 9 xited with Gillian. I was a bit blue, and being blue always makes
10 an. I was a bit blue, and being blue always makes me satirical, s
11  else, and I said it out of the blue, but in my mind it was as if
12 ly have helped that I was a bit blue. The fact that they reserved
13 sessed. Tonight he wore a faded blue polka-dot cravat at his thro
14  straight nose and bright, pale blue eyes. She had never known an
15 indows smouldering. It was that blue and red, that blue especiall
16  It was that blue and red, that blue especially, I felt in need o
17  better. Then, quite out of the blue, it is discovered by one of
18 avelling through the air like a blue flame, killing her victims s
19 d clothes. Spitting that Maid's blue flame right in the face and
20  had to unfreeze the pipes with blue-flames. In the beginning we

                                                          (FECCS)
```

This kind of a concordance is called a **KWIC** concordance (*Glossary of corpus linguistics*). KWIC stands for Key Word In Context, which means that the search word is in the center of the line, with more or less context around it. A concordance can usually be sorted in several ways. According to Sinclair (1991:33) the sorting that is often most useful is alphabetical ordering to the right of the central word. The following example is the same concordance as in example 3, but now sorted by the word to the right of the central word. The example sentences are not the same, because this is not the full concordance but only part of it, and the sorting has changed the order of the sentences. The example starts from sentence number 16; i.e. it is not taken quite from the beginning of the concordance.

Example 4: Concordance, search word *blue* (sorted 1st word to right)

```
16 f-naked body. Now she takes the blue bottle and moistens a corner
17 ly away and reaches for another blue bottle. That has a goose qui
18 mirror and one of the minuscule blue bottles, stoppered with a co
19 ped pine of the dresser and the blue bowl of orange marigolds and
20  else, and I said it out of the blue, but in my mind it was as if
21 d wear the distinctive electric-blue cap bands, shoulder boards a
22  woman, was resplendent in pale-blue chiffon and cloth of silver,
23 e nephew fills his glass from a blue-and-white china decanter of
24 mdash; she was wearing a bright blue dress from the early days of
25 atalie. She had put on her best blue dress with the wide white co
26  It was that blue and red, that blue especially, I felt in need o
27 ky and flaxen-haired, with cold blue eyes and a sharp chin. I had
28 ely pale oval madonna face with blue eyes and her hair was light-
29  continue straining your lovely blue eyes and end up bumping into
30 trudged through icy snow. Blank blue eyes stared at the old man w
31 . And this old woman's freakish blue eyes had turned it into some
32 t-chested and had pale skin and blue eyes and long fingers. Her h
33 e? The gentle pity in the faded blue eyes robbed Mattie of the an
34 ell, but at least the prominent blue eyes had not been veiled. Th
35 s to them, even when his vacant blue eyes were on Farthing and he
36 owever, he stared down with his blue eyes at the old table just b
37 oyish looks, fluffy blond hair, blue eyes and shy smile, Billy wa
38 he colour of her somewhat blank blue eyes &dash; I tell you, litt
39 in the metropolis. But his baby blue eyes missed very little. Bef
40 me through them, which made her blue eyes look huge. "You could n
```

(FECCS)

It can now be seen that this kind of sorting brings out words that the central word most often occurs with. In this short concordance it seems clear that *blue* is quite often used in the context of *blue eyes*. Although this concordance was not sorted by the word to the left from the central word, it can be easily noticed

that *blue eyes* are often further characterized here by some other adjective, in these examples *lovely, faded, blank, prominent, freakish* and *baby blue*. The full concordance of *blue* sorted by the first word to the right is given in Appendix 1.

Concordances can also be used to count word frequencies. Frequency lists can usually be arranged in the order in which the words occur in the text, alphabetically or by their frequency, with the most frequent words first (Sinclair 1991:29-32). Such lists tell something about both languages in general and about the text under scrutiny. If a frequency list is made from a comparatively short text, e.g. one chapter of a book, it can be quite easily seen what the text is about by examining the words that occur most often.

1.2.2.2 Collocation

Collocation means the co-occurrence of words in a text. *Glossary of corpus linguistics* defines a **collocate** "a word which occurs in close proximity to a word under investigation". In the case of collocation, the central word is called the **node**. Below, we have again a short extract from the previous concordance:

Example 5: Collocation

```
 92  He wore dark corduroys, a navy blue pea jacket, and a fisherman'
 93  wearing grey slacks and a dark-blue plaid shirt, packing our foo
 94 sessed. Tonight he wore a faded blue polka-dot cravat at his thro
 95 an sprint to a large silver and blue Road King, which he enters t
 96 dress of dark blue satin, and a blue rose at the apex of her deco
 97 herself between a lady in royal-blue ruffles with a fan and casta
 98  if she could come get the navy blue rug from the dining room. "N
 99 rug from the dining room. "Navy blue rug," Macon repeated. (He wa
100 wn her chin, it was matting her blue sailor dress, blood, oh dear
101 ess and wearing a dress of dark blue satin, and a blue rose at th
102 at was the Countess of Powis in blue satin, diamond-embroidered;

                                                          (FECCS)
```

Blue is the node word; all the words surrounding it are its collocates. Usually, four or five words before and after the node are considered relevant. Words further away from the node are usually too far away to be closely connected with it in meaning.

Collocates tell what kinds of contexts the node word most often occurs in. In both examples 4 and 5 it can be seen that *blue*, aside from eyes,

often seems to be used in connection with clothes and fabrics (*jacket, shirt, dress, rug, satin*). The full concordance in Appendix 1 also points out *sky* as a frequent collocate of *blue*.

1.2.2.3 Lemma

A **lemma** is the composite set of grammatical word forms. For example, the word *to give* has the distinct forms *give, gives, given, gave, giving* and *to give*. These words together are the lemma of the word *to give*. Arranging words into a lemma is called **lemmatization** (*Glossary of Corpus Linguistics*). Lemmas can be useful when, for example, a researcher wants to see the concordance for the word *to give* in all its word forms. In a language like Finnish, where words are highly inflected, lemmas can prove to be very useful in corpus research.

However, one can also argue against the lemmatization of a corpus. Both *Glossary of Corpus Linguistics* and Sinclair (1991:41-42) note that there are reasons why it should not be done. Since the computer cannot lemmatize words automatically, it is sometimes the researcher's personal judgment that decides which words are related and which ones are not. There can be also differences between the word-forms, for example their collocates can be very different. That is, the different word-forms of the same word occur in completely different environments, although they can be considered to be the inflections of the same word. For example, the word *blue* ('a colour') and the word which looks like its plural form *blues* ('a style of music') should not really be grouped together for most purposes.

This chapter has introduced some of the key concepts in corpus linguistics. Annotation and alignment are important in connection with the compilation of a corpus, whereas concordances, collocations and lemmas must be understood when one wants to start to use a corpus. Before proceeding to the uses of corpora, there is one more thing to discuss. In order to do anything with a corpus, a researcher must have a program that is able to browse and search the texts. The following chapter will shortly introduce three programs that can be used to deal with corpora. There are, of course, plenty of corpus software available in the world; the programs in the following chapter have been

selected here because they have been used at the Department of English, University of Jyväskylä, and because they serve as good examples of the basic principles of how corpus software works.

1.3 Examples of corpus programs

There are plenty of programs for browsing different kinds of corpora. Some programs can handle almost any kind of text, whereas others have been made for a specific purpose and require their own data format.

Effective processing of different types of corpora requires programs that are meant specifically for the relevant kind of corpora. For example, although the texts in a translation corpus can be separately processed in a program that is meant for only one language, translations can be properly studied only in a program that can handle both languages at the same time. Correspondingly, even though many annotated corpora can be processed in any concordancer that knows how to hide the tags, the full advantage of annotated corpora can only be gained with programs that can use the annotation properly. For instance, the program should be able to make searches where the tags are used as search parametres (e.g. to retrieve only those instances of *will* when it is used as a modal auxiliary).

Three programs are introduced in this chapter. The first two of them, MicroConcord and WordSmith Tools, are for monolingual corpora and for very general purposes, that is, they can be used to read almost any text. The third program, TCE, has been specifically made for the kind of translation corpora that have been compiled in Norway, Sweden and Finland. These programs were selected to be introduced here for a few simple reasons: they can be used to process the corpora that have been compiled at the English department at the University of Jyväskylä, and they were used to produce most of the material and examples of the present paper. In addition, especially the first two are typical examples of corpus software.

It might be noted here that although the following sections only mention that the programs search for "words", the searches can be defined more specifically than by a single word. For example, each program can make a search for *book**, where the use of the asterisk results in all words beginning with *book* (e.g. *books, booking*) to be found. In addition, all programs have their own more exact ways of defining search strings. They will not be examined here in detail. Suffice it to say that most corpus browsers allow for quite complex search strings to be entered.

1.3.1 MicroConcord

MicroConcord was published by Oxford University Press in 1993. It is a fairly simple, DOS-based program that searches for words and makes concordances. Concordances can be sorted by the node word, and by 3 words to the left or to the right of the node. It can also count the frequencies of the most common collocates of a word. MicroConcord is suitable for examining texts in one language only. It comes with its own small database, which spans several text types.

MicroConcord can, however, read any text that is in ASCII format. No special tagging is needed, and it is not language specific either. Therefore, although simple, it is rather useful. Basically, a researcher can take any stretch of text and examine it in MicroConcord. There is the limitation, though, that as a DOS program MicroConcord does not support other than the standard character set, plus some accented characters. Character set problems will be addressed in chapter 2.2.

1.3.2 WordSmith Tools

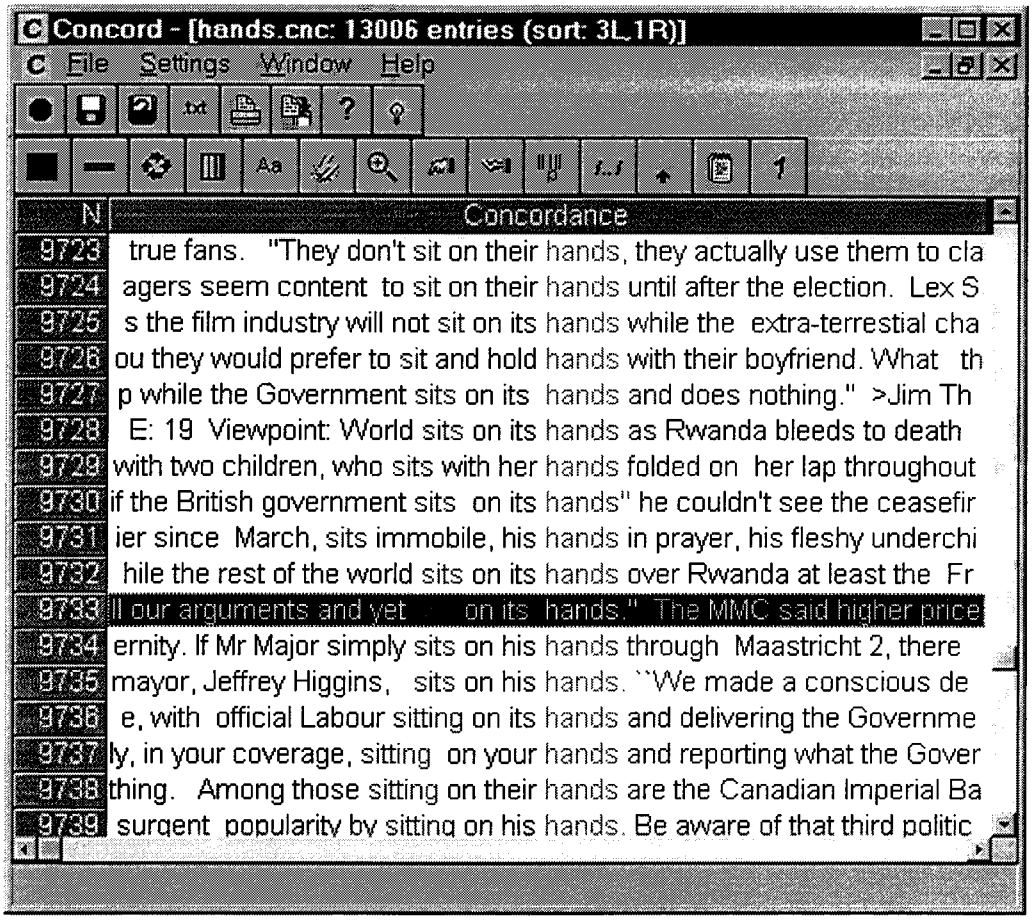
WordSmith Tools is a Windows program that is a more advanced version of MicroConcord. It has basically three parts: the Wordlist tool, the Concord tool, and the KeyWords tool. A typical scene from WordSmith Tools can be seen in picture 1.

The Wordlist tool makes lists of all the words in a text, both alphabetically and in the order of frequency. Concord works in the same way as concordancers usually do, that is, it shows words with some amount of context around them. It also allows sorting and has rather good options for handling collocates. KeyWords searches for what are called key words. These are words that have an unusually high frequency in a text. WordSmith Tools also has an alignment tool, even though the program is mostly used for one language at a time, and a few other functions.

Like MicroConcord, WordSmith Tools also reads any ASCII text that is given to it. In addition, it also recognizes SGML tags and is able to ignore them, which is a useful feature, strange as it may seem. It enables WordSmith Tools to read files that have been tagged for some other program, and still

make sense out of them. Examples 3-5 in the previous chapter, in connection with concordance and collocation, were printed from WordSmith Tools.

More information about WordSmith Tools can be found at http://www1.oup.co.uk/elt/catalogu/multimed/4589846/4589846.html. The program can also be downloaded through the Web.



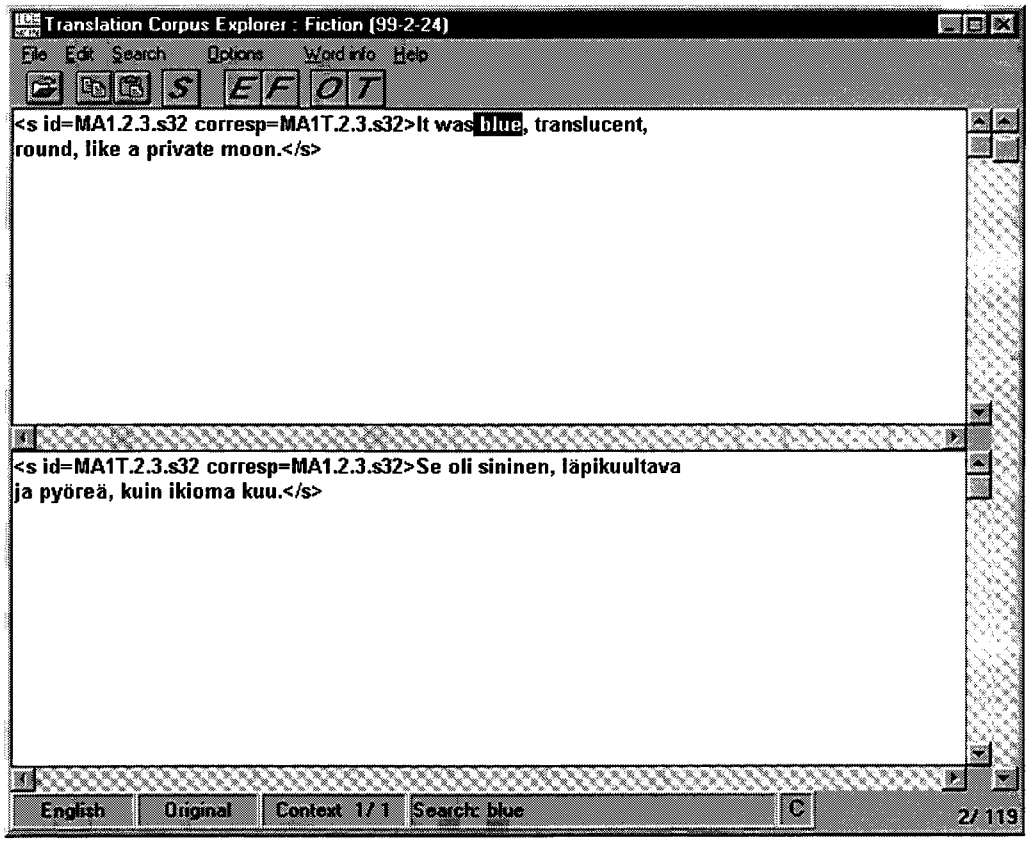
Picture 1: WordSmith Tools.

(http://www1.oup.co.uk/elt/catalogu/multimed/4589846/screen1/text1.html)

1.3.3 TCE (Translation Corpus Explorer)

Translation Corpus Explorer, or TCE, has been developed by Jarle Ebeling at the University of Oslo. It was originally made for the ENPC (English-Norwegian Parallel Corpus) project, but has been later used in Sweden and Finland for other translation corpora, for example FECCS.

As the name implies, TCE is made especially for examining texts and their translations. The program can search for a word, and the results are displayed in two windows. The upper window contains a sentence in which the search word was found, and the lower window displays the corresponding sentence in the other language. Below is a typical screen in TCE:



Picture 2: TCE

The following is an example of the printout of TCE. When printed on paper, the examples are printed simply following each other, as shown below:

Example 6: TCE printout

```
==============================
<s id=WB1.2.s141 corresp=WB1T.2.s142>Tonight he wore a faded
<b>blue</b> polka-dot cravat at his throat which set off his
tan admirably.</s>
------------------------------
<s id=WB1T.2.s142 corresp=WB1.2.s141>Sinä iltana hänellä oli
kaulassaan haalistunut sinipilkullinen solmio, joka toi
rusketuksen esiin suorastaan ihailtavasti.</s>
==============================
<s id=WB1.2.3.s27 corresp=WB1T.2.3.s28>  He had a long,
straight nose and bright, <b>pale</b> blue eyes.</s>
------------------------------
<s id=WB1T.2.3.s28 corresp=WB1.2.3.s27>  John Clearwaterilla
oli pitkä suora nenä ja kirkkaat vaaleansiniset silmät.</s>
==============================

                                                    (FECCS)
```

TCE requires that the texts are not only tagged and aligned, but also made into a database from which words can be searched easily. This means that the final database can be processed by TCE only, and not by any other program. There is also a version of TCE that can handle more than two languages at the same time. The full TCE printout of search word *blue* is in Appendix 2.

This short chapter has briefly introduced three programs that can be used to browse corpora. Two of them, MicroConcord and WordSmith Tools, are made to deal mainly with monolingual corpora, whereas TCE is made specifically for certain kind of translation corpora. The following chapter goes on to explain what kind of things corpora are actually used for.

1.4 Uses of a corpus

This chapter introduces some of the uses of corpora. The point here, especially in the first section, is to consider them without much reference to any specific types of corpora. Rather, the aim of this chapter is to give a general idea of the advantages that can be gained by using any corpora. Part Three of the present paper will give more detailed examples of the use of different types of annotated corpora.

Generally, computer corpora are used for different kinds of linguistic research. They can also be utilized in teaching and in many automatic procedures which involve language, such as machine translation and many applications that concern artificial intelligence. Different types of corpora tend to have different uses. There are, however, some advantages in using a corpus that are common to all kinds of research and all kinds of corpora. They will be dealt with here first. After that, some of the more specific uses will be discussed.

1.4.1 General advantages of corpora

The most obvious advantage that any corpus offers is the vast amount of material available. Computer corpora enable the kind of research that could not be done manually, research that involves text material consisting of millions of words. It would take a lifetime for a single researcher to go through such quantities of material manually searching for a word or pattern, whereas a computer can do it in a few seconds.

Aside from that, the most often emphasized advantage offered by corpora is that a corpus provides a researcher with hundreds, even thousands of examples of words in their actual use. A corpus can bring up phenomena in texts that might not be noticed otherwise. It brings out patterns from a text and displays words in their real context, covering all instances.

Thus a corpus search reveals examples and uses that might not have been noticed otherwise, or that would have been ignored as irrelevant, and forces researchers to question beliefs that they may have taken for granted. A corpus can drastically alter the ideas that people have about language. By

providing evidence of how language is actually used, corpora can effectively challenge people's intuitions and force them to see things that they might have neglected otherwise. Among many others, Sinclair (1991:4) notes that there is a huge difference between what people think about language and what corpus evidence proves. Some examples are given here to clarify this point.

For example, the word *back* is usually taken to mean basically the human body part. When asked, that is what most people would say. Most dictionaries rank it as the most important meaning for the word, and it is the meaning that is given first in the dictionary. However, corpus evidence proves that *back* is most commonly used in the meaning "in, to, or towards the original starting point, place or condition", which dictionaries usually do not consider very important usage (Sinclair 1991:112). This information is important, for example when teaching English as a foreign language. Although dictionaries and maybe school textbooks, too, are likely to explain *back* as a part of the human body, outside the classroom the students are much more likely to see it in its adverbial sense.

Same kinds of cases can be found in Finnish as well. For example, many people might say that the Finnish word *tosi* essentially refers to the truthfulness of something. However, a simple search from the FECCS corpus does not support this at all; mostly, *tosi* is used to intensify the meaning of other words, like *tosi kova, tosi kauan, tosi tehokas, tosi mukavaa.*

As Stubbs (1994:218) puts it, "computers make it more difficult to overlook inconvenient instances". A corpus forces the researcher to see how language really is used, not only how researchers think it is used. Of course, a corpus may not only challenge people's thoughts, it can also confirm them. Leech and Fligelstone (1992:122) mention that corpora are nowadays widely used to test hypotheses and provide evidence of language use. Since there is a vast amount of material in any corpus, the results of corpus searches must be taken seriously.

What a researcher has to keep in mind, however, is that although a corpus can provide lots of evidence, certain reserve is needed. As Sinclair (1991:5) says, there are "plenty of bizarre and unrepresentative instances in any corpus". There may be, for example, a strange sentence that represents only the

usage of one author in a certain unusual context, not the language as a whole. This is where the quantity of corpus evidence really comes in. If a strange collocation for a word occurs only once in a corpus of a hundred million words, it is not likely to deserve much notice. If it comes out of a corpus of a million words, however, its importance cannot be decided offhandedly. It may be said that good judgment and, after all, intuition, must be used to decide which cases really are representative of the language and which are not.

1.4.2 More specific uses for corpora

Different corpora can be used for several different kinds of linguistic research. The type of research depends on what kind of corpus one has available, e.g. how large it is and what kind of material is in it. It is not within the scope of the present paper to examine the uses of different kinds of corpora exhaustively; rather, a general picture and individual examples will be given here.

One common use for corpora is the making of dictionaries (Leech and Fligelstone 1992:120, 123, Sinclair 1991:1-3). The Collins Cobuild dictionary was mentioned earlier in the present paper; in addition, at least two other publishers, Longman and Oxford University Press, have worked on corpus-based dictionaries (Leech and Fligelstone 1992:120). The advantages of this kind of dictionary building are obvious. The compilers need not have to try to come up with proper examples of language usage themselves, since real-life examples can be retrieved from the corpus. It is just as easy to find new words as to judge their importance by their frequencies (*Bank of English – Questions and Answers*). For instance, if a word is a highly technical term which a person who is not an expert of the field is never likely to come across, it may be left out of a general-purpose dictionary. However, a corpus may reveal that a technical-sounding word actually occurs quite often in newspapers and magazines, and thus warrant its inclusion in the dictionary.

Leech and Fligelstone (1992:123) also mention that a dictionary need not be in a printed form. The idea of a dictionary that could be searched and updated by the computer is something that is becoming more and more popular. This is close to what Sinclair (1991:24-26) calls a monitor corpus, a

corpus which is used to filter out new words and meanings from constantly changing and renewing material.

A translation corpus provides information not only of one language but of at least two languages at the same time. In addition to that, the texts in the two languages are translation equivalents. Such a corpus is naturally ideal for studying translations and helping the translation process. However, a translation corpus can as well provide important information about one of the languages only. One can compare, for example, Finnish original texts and texts that have been translated into Finnish from English. Such research within a language points out how translations are different from original texts. For example, sentence structures in translations are not always the same that occur in original texts in that language. Further, translated language often includes words that would not be used in original writings.

Corpora can also be used to support language learning. A corpus can provide evidence about which structures in reality are the most common and the learning of which should thus be emphasized. For example, Biber et al. (1994:171-174) compared several pedagogic grammar books to corpus evidence. Their research showed that the structures which the grammar books emphasized were actually quite rarely used in the corpus. Some very common linguistic constructions, however, were overlooked in the grammars. Grammar books often opt to teach structures that are the easiest to acquire and easy to teach in the classroom. However, some of the harder structures are likelier to occur in actual situations of language use. (Biber et al 1994:171-174).

Sentences exemplifying correct usage can also be taken from a corpus. Often examples in textbooks, although grammatically correct, do not sound quite natural. Although invented example sentences can always be guaranteed to be structurally correct, it may be hard to come up with situations in which someone would really use them, while all examples taken from a corpus have been used in real text or speech. Therefore they give a more realistic idea of the uses of a word or a structure.

Corpora can also be used to study learner language and to identify where learners' errors originate. One can compare learners' language with the language of native speakers as well as compare the errors of learners from

several different countries (Granger 1996:43-49). Thus one may identify errors which are due to transfer and caused by the learners' mother tongue, and those which are common to all language learners.

A few more uses for corpora may be mentioned here. Leech and Fligelstone (1992:121-124) give several examples of different kinds of research areas where corpora can be utilized. Spoken language corpora can be used, for example, to develop speech synthesis and speech recognition. Machine translation can be developed through the use of corpora. Text checkers, that is, programs which evaluate writing and correct spelling and grammar, can also get valuable information from corpora.

This chapter has given a general picture of the tasks that corpora can be used for. First, I discussed the advantages that are to be gained from the use of any kind of a corpus. Secondly, examples of more specific uses for corpora were given. Note that the list here is only a list of examples: it is by no means all-inclusive. There are many more applications for corpora, both in the academic and the commercial world. The purpose of this chapter is only to give a general introduction of the possibilities that corpora offer.

The following chapter is about a topic that is strictly speaking out of bounds of the scope of the present paper, but so important that it must be mentioned nevertheless. After one has decided to compile a corpus and has, perhaps, already considered specific texts that could be in it, one must check whether one needs a permission to include the texts into a corpus. The following chapter briefly deals with questions relating to the matter of copyrights.

1.5 Copyrights

One issue that needs to be dealt with briefly is copyrights. It is, indeed, a problem of its own, and the scope of the present paper does not make it possible to deal with it in detail. However, a few words about copyrights are necessary.

In order to include any texts in a corpus, one must have the permissions for it from the copyright holders. Since the whole idea of having electronic texts is quite new, there may be problems with getting the permissions. The copyright laws have not changed fast enough to take electronic text into account, and much of the traditional terminology used in connection of copyrights is not clearly defined with computerized material. For example, as Hockey (1998:105) points out, it is not clear what is a "new edition" in case of electronic texts, since they can be changed easily all the time. There is also the issue of "making copies" of the material: the word "copyright" literally means the right to make copies of something. In case of the printed word the meaning is clear, but how should one define making a copy when dealing with computers? Every time a file is saved on a disk, moved to a different computer, or even saved as a temporary file by the computer, a new copy is made.

Hockey (1998:105) also notes that there have been problems because researchers do not know where to ask for the permissions, or because the copyright holders do not know how to answer the requests. As the whole issue is rather new, many publishers may not yet have any previous experiences or standard practices for dealing with such requests. It may be also that neither the researchers nor the copyright holders have a clear idea of what kinds of rights they can request or retain, and what kinds of rights one actually needs in order to process and use the corpus. As Hockey (1998:105) also points out, copyright laws are different in different countries. This can prove a problem in international projects.

Since the laws are not quite clear here, it may be that some publishers take a strict principle and never give any permissions to convert the texts into electronic form, or let their material be used in corpora. A new trend also seems to be that some publishers give permissions for a limited time, for

example for two or three years. Some may also demand payment for the right to use the texts, even for linguistic research.

In any case, anyone who decides to compile a corpus should find out about the relevant copyright issues and get the necessary permissions for the material. The whole matter is likely to become clearer soon, since the need for such permissions has become greater during the last few years. Hopefully, there will also be some international guidelines that will make the compilation of multilingual corpora easier and give rise to further international projects. There have already been many plans towards international laws and practices. For example the European Union has published a report called the Green Paper on Copyright and Related Rights in the Information Society (available, for example, at http://www.ispo.cec.be/infosoc/legreg/com95382.doc in Word format).

Note that copyright laws also apply to non-commercial texts. For instance, essays written by elementary school students cannot be included in any corpus without proper permissions. The copyright laws about this may be somewhat different in different countries, but at least in Finland it is worth noting that a school cannot give a permission for the use of the essays automatically; the essays are the intellectual property of the authors, i.e. the students, and the permissions must be obtained from the students themselves (their age is not an issue here). The copyright is always held by the author, unless it has been explicitly moved to someone else.

The only types of texts that can be included without permission are older texts which are not protected by copyright laws anymore, and texts that were not copyrighted in the first place, such as folklore, some legal texts and many religious writings. They may not, however, provide the basis for many different kinds of research projects. This point will be briefly taken up again in chapter 3.2.3, which discusses text archives.

1.6 Summary

The first part of the present paper has been an introduction to computer corpora. In it, I have tried to create a basis from which the reader can go on to read about the more specific issue of compiling computer corpora. The introduction can also be seen as a general introduction to computer corpora for someone who is not familiar with the field beforehand.

Chapter 1.1 explained what computer corpora are and what different types of corpora exist. It gave a framework to classify different types of corpora and explained about some of the characteristics of certain types of corpora. The classification I presented here is not the only possible way to look at them; indeed, corpora can be classified according to many other features, too, if need be.

Chapter 1.2 introduced terminology that is relevant to corpus linguistics. In it I briefly explained terminology that concerns the compilation of corpora, and the use of corpora in general. After reading the chapter, the reader should have a general idea of central concept in corpus linguistics.

Chapter 1.3 presented three programs that can be used to search computer corpora. The programs were the ones that have been used at the English department at the University of Jyväskylä, and they also serve as good examples of how corpus software usually works.

In chapter 1.4 I explained what computer corpora can be used for. Advantages of all corpora and more specific uses of corpora were given as examples. It may be noted here again that the purpose of the chapter is only to give a general idea of the uses of corpora; in addition to the examples given here, there are many other uses, too.

Chapter 1.5 dealt with the issue of copyrights, which is fairly important for corpus compilers. Before any texts are included in a corpus, suitable permissions must be acquired from the copyright holders. Although the issue was not discussed here extensively, its importance should not be underestimated.

Part Two of the present paper will consider computer-related issues. Since language corpora are nowadays kept in computers, there are decisions to be made about the equipment to be used. There are also many questions and

problems that may come up in the compilation and use of corpora. These concern, for example, character sets, file formats, and the different ways to store information and take backup copies. The following part of the present paper will deal with these.

2 COMPUTER ISSUES

In order to compile and use a corpus, one must have a computer with which to process the data, and suitable programs for the purpose. This part of the present paper deals with issues that have to do with computers and related matters. The purpose is to give the reader some idea of what kinds of options there are concerning the hardware and software and what kinds of problems one can encounter with them, and in general, give information about hardware and software that is relevant with computer corpora. The purpose is to help the reader choose hardware and software that would be the best possible for the corpus and for its users.

Since the development of computing is very fast, it is, in many cases, impossible to give very detailed recommendations. Rather, some general guidelines in computer matters can be given. Although both the hardware and software change rather quickly, some basic principles tend to apply for longer periods of time. There is also the fact that the selection of hardware and software depends on the type of the corpus, and the research needs of the users. A small corpus for a small research project which only needs concordances, has somewhat different needs than a multi-million-word corpus with many users, who need different types of statistical processing of the data. This chapter cannot give any specific advice for a specific type of a corpus; rather, it has to give advice that *usually* applies to *some aspect* of corpus use or compilation.

One should also be aware of the fact that although the state-of-the-art of computers changes all the time, corpora are not usually very demanding. Much of older hardware and software is still quite suitable for work with corpora. This has to do, to a large extent, with the nature of corpus data. Textual data rarely takes much disk space and searching for a word does not usually take huge amounts of time, depending, of course, on what else can be done with the program. The most advanced machines available often offer better performance with demanding mathematical calculations, and with tasks that involve graphics or sound, such as 3D modelling or music editing. The manipulation of corpora, however, is mostly about dealing with strings of text.

Although newer machines can undoubtedly do it faster, older machines are quite adept at it, too.

On the other hand, this part of the present paper is also tied to the hardware and software available at the time of writing. For example, the operating systems and storage options introduced here are the options worth considering at the time of writing; in a few years, however, some of them may be obsolete, or some of them may have become the new industry leaders. The purpose for introducing such systems here is because, even if the most advanced users will discard them in a matter of years, there will be, on the other hand, plenty of other users who will stick to their old machines for years to come. As was stated in the previous paragraph, corpora do not necessarily require the best that is available, and it may be, indeed, that all the users of the corpus have rather old equipment, too. Therefore, even some of the more detailed information here may be of use to the compiler and user of a corpus for more than a couple of years. Even when both the hardware and software have completely changed from the present, the reader can hopefully see the general principles that apply to the selection of the computing environment.

This part of the present paper is not meant for computer experts. They are already likely to know everything that is said here, and will probably find the contents of these chapters rather tedious. The information here is meant for a reader who has already used computers, who understands basic things about them (such as what is a *file* or a *directory*), and who has some experience of using computer programs and knows their basic functions (such as cutting and pasting text in word processing programs). It is also expected that the reader is at least somewhat familiar with e-mail and the World Wide Web. No extensive knowledge of computers is needed, however; the readers are expected to be linguists rather than computer wizards. The purpose of these chapters is to help them gain such information about computers that is relevant when working with corpora.

It may be noted here that there are bound to be many computer terms that are mentioned in the text but not explained. These might include references to *bits* and *bytes*, or a remark about *client/server* –networks. In many cases, these have to do with details that can be quite safely skipped, if the

reader feels that he or she does not understand them. The important point here is to understand the main principles; the details, shallow though they are, are for those who find them interesting. If it is necessary to understand a term, it is also explained here. Most of the vocabulary used can be found in any dictionary of computer terminology, so it would not serve the purpose of this paper to explain them here.

Chapter 2.1 describes hardware and operating systems. It introduces some criteria for choosing hardware and the operating system, and also discusses some operating systems in more detail. Chapter 2.2 is about character sets and related issues. It includes descriptions of character sets, plain text formats, and problems that relate to these, especially when moving files from a computer to another. Chapter 2.3 introduces options for storing the corpus data and gives general advice on backup copies and storing the data. Chapter 2.4 is about software: what is required of corpus software, and how to get suitable programs. Hopefully, all this will give the readers a good basis for acquisition of hardware and software for corpus compilation and use, and prepare them for the problems that may come up with computer related issues.

2.1 Hardware and Operating systems

This chapter deals with different operating systems, their qualities, their advantages and their disadvantages when compiling and using a corpus. It also briefly considers criteria for choosing hardware. As mentioned above, this chapter is written for a person who does not have an extensive knowledge of computers, but rather, some rudimentary knowledge of how computers work and how to use them. People with more experience with computers may not find this chapter very relevant to them; the whole question of choosing an operating system or hardware may not seem relevant to one who is used to using different systems, moving files around between them and configuring programs for one's personal use.

However, people who compile and use a corpus are often no computer experts but researchers whose interests lie elsewhere than in the technology. For them the computer is just a tool to achieve certain results: the tool itself is not interesting. From their perspective, the question of different operating systems and hardware is relevant only because this information makes the compilation and use of the corpus as smooth, easy and efficient as possible.

That, indeed, is the key reason for the existence of this chapter: to help make useful decisions regarding the operating system. It must be said, however, that in many cases the compiler of a corpus may not have much choice regarding the operating system. Often, indeed, it may be the case that there are only a few computers at hand, with the operating system that happens to be in them. In such a case, of course, the question of different operating systems and hardware is a bit irrelevant; one just has to get along with whatever is available. On the other hand, it may be that it is decided from the onset of the project which programs are to be used with the corpus; in such a case, there is probably no choice whatsoever, because one must use the operating system that the programs happen to work in, and get the hardware that both the operating system and the corpus programs require.

However, if it has not been decided which corpus browser to use, the choice of the operating system and hardware is relevant. For example, simple concordancers are available for a variety of operating systems. Also, if you plan to make the program yourself or have someone make it specifically for

your corpus, the operating system needs to be decided. Of course, even in that case the existing hardware may limit the choice.

Yet another reason for considering the choice of the operating system and hardware is the fast development of computers. To avoid the need to completely recompile the corpus after present systems are outdated, and to make the corpus last for more than a few years, the choice of hardware and software needs to be considered. These questions are also related to other things in addition to operating systems, such as programs, file formats and character sets, but these, in turn, are very much related to the operating system and hardware.

In any case, there are situations where the researcher is free to choose both hardware and software, and in that case, it is worthwhile to consider their pros and cons. The following section presents some criteria for choosing the hardware. Because computers change and develop quite fast, it would not be wise to give any concrete suggestions about the type of computer; however, there are general guidelines that apply now, and will apply in the future, too, for choosing useful hardware.

2.1.1 Criteria for choosing hardware

There are many details to consider when choosing suitable hardware. Antworth & Valentine (1998) give some useful suggestions for getting the right kind of machine.

Their first point is that the hardware should meet the needs of the software (Antworth & Valentine 1998:171). There is no reason to buy a very expensive or popular model if the software does not require it. One should avoid the kind of "my-computer-is-better-than-yours" –arguments that computer aficionados tend to get into: no type of computer is automatically better than another one just because someone claims so. The best computer for your needs is the one that runs your programs and that you can use comfortably. Antworth & Valentine (1998:171) also note that you should buy the kind of computer that you need now, not something that you think you will need later on. They give the following example: you should not buy a $4000 computer now, if you do not need it, no matter how new and powerful it is;

instead, buy a $2000 machine that you need. You can buy the $4000 computer two years later for $2000. The end result is that you have spent the same amount of money, but you have two machines instead of one.

You should also consider what kind of computers other people in your local computing community use (Antworth & Valentine 1998:171). If everyone has different kinds of computers, it is hard to share data and expertise with others. There should also be maintenance, such as repair and technical support, available for the kind of computer you have (Antworth & Valentine 1998:171-172).

Antworth & Valentine (1998:171) also point out that the computer should support a character set appropriate to the language you are studying. Languages that use a completely different type of script, such as Russian and Greek or, even more so, Chinese and Japanese, need a character set of their own. This, however, is not a huge problem nowadays, because the graphical user interfaces of modern personal computers allow for plenty of different types of fonts to be installed and used at the same time.

Very different types of languages, however, may even require consideration for a new keyboard that is made for them. Note that even in the western countries that use the Roman script there are local differences to keyboards. For example, the Finnish keyboard has the characters *ä, ö, å, Ä, Ö,* and *Å* that are missing from English and American keyboards. A few letters difference of course does not make using the keyboard impossible, and even bigger differences can be dealt with. For example, those who write in Russian have little stickers with the alphabets on them which can be glued on a normal western keyboard, thus making the writing in the Russian font easier. For far different languages, however, a separate keyboard in addition to the font may be needed.

Antworth & Valentine (1998:172) also say a few words about portable computers. They suggest that one should only buy a portable computer if one genuinely needs it. They are more expensive, and have rather poor screens and keyboards compared to desktop computers. If you travel a lot or otherwise need to use a computer in many different places, a portable computer is a fine choice; if not, rather get a cheaper, but more effective desktop model.

Depending on what kind of a computer you have got, you may have a choice of the operating system. Some computers, like Macintosh, come with their own operating system which is not usually changed (except updated); some others, like IBM-PCs, offer more choices. The following section lists some of the things about corpora that one should remember when making the choice.

2.1.2 Questions to consider when choosing the operating system

There are certain situations when the choice of the operating system is particularly relevant. They are briefly listed in the following paragraphs. Note that no answers to the problems are offered here, because there are no foolproof answers to any of them. Rather, the different advantages and disadvantages of different operating systems are dealt with in sections 2.1.3-2.1.5, and the readers may draw their own conclusions on which is the best alternative for them. This section of this chapter only tries to point out questions that need to be considered and situations where the choice of the operating system is particularly important.

1) If the corpus is going to be used for a long time, it is not irrelevant which operating system it is in. Operating systems do not only receive regular updates, but with time, some of them are forgotten and others come along. The programs used for the corpus may exist only for that particular system, and because of that it may be more or less impossible to move the corpus to another system and still be able to use it. Even if the corpus files are in such a format that they can be used in other computers and programs, it takes extra time for the users to familiarize themselves with new software.
2) Corpora may also be very large. In such a case, moving them anywhere is more or less problematic and time-consuming, and different operating systems only make it worse. If there is even one simple file format conversion to be made, it is not laughing matter if you have 10 gigabytes of text to convert. Of course it can be dealt with, but it takes time, and no-one wants to be doing it too often.

3) A problematic situation also comes along when there are plenty of users for the corpus. There may be plenty of users to begin with, or with time, more people may emerge who want to use the corpus. These people may have a plethora of different kinds of computers and operating systems, some of which did not even exist when the corpus was compiled. A compromise must be found that best pleases everyone.
4) If you plan on programming the corpus browser or any related programs yourself, you must decide fairly soon which operating system it is going to work in. Parts of program code, especially the parts concerning the user interface, are system-dependent and may have to be reprogrammed to a large extent if the same program is converted for another operating system. Depending on what kind of a programming language is used, it may not be possible to convert the program for other systems at all.
5) It must be recognized that there are different types and levels of users. Even though a certain operating system might theoretically be the best for a certain kind of corpus, it may be that none of the users can use it or are willing to learn it. Beginners, ordinary users and experts often prefer different operating systems. It may be said that none of the widely used operating systems are completely good or bad: any of them is good for a certain group of users, for certain types of tasks. The following chapters will, indeed, try to deal with the pros and cons of different operating systems, not only considering their suitability to corpus processing, but also considering their suitability for different kinds of users.

It must be noted here that the question of operating systems is tied to the question of programs, and file formats. On one hand, there are programs that are available for several operating systems and use the same file format, no matter which system it is in. In such a case, it may not make any difference which operating system is chosen for compiling and using the corpus, and the users' preferences dictate the choice. On the other hand, some programs are available for only one system, and some file formats may be somewhat system-specific. For example, plain text formats tend to be a bit different in different

systems. Chapter 2.2 will look more closely at the problems with different character sets and plain text formats.

Of course, in any case it saves time and trouble if the material can be kept in the same operating system and in the same computer most of the time. If it has to be moved, the easiest case to handle is when the file format can always be the same, so that both during the compilation of the corpus and over longer periods of time the corpus never has to be converted to a different format. How often this is possible, is of course an issue of its own; new programs often come along, which will require changes or additions to the old data.

In many cases the choice of operating system depends totally on the program that will be used to browse the corpus. If it has been decided in the beginning that a certain program will be used, in many ways it is easiest to compile, e.g. scan and proofread, the material in that operating system, too. In that case the files do not have to be moved around, or at least there will be no unprecedented problems when moving them and trying to get them to work in another machine. Such a procedure also helps to guarantee that the character set and file formats used are compatible at all stages of the compilation and use of the corpus.

However, this may create some problems from the users' and compilers' point of view. For example, if the corpus is going to be used in Unix, but none of the people who proofread the texts have any experience of using it, it has to be considered whether it is useful to teach people the use of a new operating system and word processor, or to process the texts in another operating system first, then move them over to Unix.

The following chapters briefly introduce some of the most common operating systems. It may be noted here that all information in this chapter is relevant at the time of writing. However, as is well known, computers tend to develop rather quickly, and some of the information here is likely to get outdated within a short time. The end of this chapter, however, will try to give some general guidelines which should apply for somewhat longer.

2.1.3 Windows

Windows is probably the best-known operating system of personal computers nowadays. There are different versions of Windows: the old Windows 3.1, Windows 95 and its updates, and Windows NT. Each one of them will be shortly introduced here in turn.

All versions of Windows offer a graphical user interface (GUI) which most people find easier to use than a character-based one, such as ordinary Unix or DOS. Windows was designed to be easy to learn and use, so that the user does not have to know anything about the internal workings of the computer to learn how to use the hardware and software (Kiianmies 1995:15-16). All Windows programs work basically in the same way, applying the same kind of graphical elements in all programs, such as windows, menus and buttons. Once the user has learned the basics of one program, it is fairly easy to get the hang of another (Kiianmies 1995:16).

Many procedures of Windows programs have been standardized. Regardless of the program, some shortcut keys, for instance, always stay the same (e.g. *ctrl-c* for copy, *ctrl-v* for paste), and the same options can be found in the same places in the menus (e.g. *exit* is always the last option in the *file*-menu). DOS programs do not have this kind of standardization; the user has to learn different ways to do the same things, depending on the program (Kiianmies 1995:17-18).

The first version of Windows, Windows 1.0, was published by MicroSoft back in 1985. It was soon followed by Windows 2.0 in 1987. Neither of them became very popular, and they were used only if needed for specific programs (Kivimäki & Rousku 1996:13-14). Compared to later versions of Windows, they were fairly simple and did not offer much to the user that other operating systems could not provide.

Windows 3.0 was published in 1990, and it became instantly extremely popular. It was updated to version 3.1 in 1992, and later on to version 3.11. These offered several error fixes and some new features to the original version (Kivimäki & Rousku 1996:14-15, Kiianmies 1995:15). Later MicroSoft published Windows for Workgroups. It is basically the same as

ordinary Windows 3.1, but it has network support, for example programs for e-mail and fax and means for file sharing (Kivimäki & Rousku 1996:15-16).

Nowadays, if any of the older versions of Windows are in use, it is likely to be either Windows 3.1 or 3.11, or Windows for Workgroups 3.1 or 3.11. To the ordinary user it usually makes no difference which particular version of these is used. Therefore, the name Windows 3.1x is often used to generally refer to all of them, meaning that the version number can be either 3.10 or 3.11.

Windows 3.1x is still widely used in older computers. It is actually not an operating system in its own right, but a graphical user interface for DOS (Kiianmies 1995:15). The operating system underneath is still DOS, and faces the same limitations, such as problems with memory.

New Windows programs are nowadays made almost invariably for Windows 95/98, and will not work in Windows 3.1x. It is possible, however, to make programs that work in both, and older programs generally do. If you want to use Windows and have only Windows 3.1x, it is likely that you can find several corpus programs for it. Any computer with Windows 3.1x in it also has DOS, which means that DOS applications can be run in it without some of the problems that come up with DOS in newer versions of Windows. None of these corpus programs are bound to be the newest ones available; however, they may include the kind of tools that you need and be perfectly suitable to the processing of the corpus that you are going to compile.

Windows 3.1x is, however, a dying operating system. In a few years it is likely that no-one will use it anymore. A corpus compiled in it should for that reason be saved in a format that can be easily moved from a system to another. It is not recommended to start compiling huge corpora in Windows 3.1x nowadays. If there is anything system-dependent in the compilation of the corpus, it will only result into problems fairly soon.

Windows 95 is probably the most common version of Windows nowadays. It was published in July 1995 and offers several new features when compared to Windows 3.1x. These include, among other things, better management of resources (e.g. memory), greater stability (programs do not freeze completely as often as before), better networking possibilities and 32-bit

applications (Kivimäki & Rousku 1996:18). The most visible change from the user's point of view is that the user interface has been remade completely (Kivimäki & Rousku 1996:18). It offers the user different possible ways to use the system: Windows 95 can be used much like old 3.1x, but Macintosh users will also find familiar procedures.

Windows 95 also makes it possible that the users need to understand as little as, or as much as they want to about the operating system. The ordinary user does not have to know much about the internal workings of the computer, whereas the experienced user can get familiar with the system on a deeper level. For instance, files can be dealt with simply by viewing them as icons in folders, with only the main part of their name visible (e.g. *my_text* instead of *my_text.doc*); on the other hand, if needed, the user can change file attributes and view full file extensions, create shortcuts to files and arrange them according to different properties.

There are several upgrades for Windows 95, named Windows 96, 97 and 98, denoting the year when they were issued. Although they do offer updates to the operating system, they are not considerably different and seem more or less the same to the ordinary user. Windows 98 is the one that is most noticably different, but even there the differences are not so big to the original Windows 95 as to confuse the users.

Windows NT is the fully 32-bit version of Windows. NT means "new technology" and it was originally designed to get rid of the limitations of DOS (Kivimäki & Rousku 1996:17). There are actually two versions of Windows NT: Windows NT server and Windows NT client (Kivimäki & Rousku 1996:24). The server version is much more expensive, but with much better network support. The client version is cheaper, but it can also work as a server in small networks.

Although Windows NT looks just like ordinary Windows, the workings underneath are very different. It was not limited in its development by any remnants of DOS, but instead was completely reprogrammed from the beginning. It is more powerful and reliable than other versions of Windows, and it offers much better network support in general. It also works in a variety of different kinds of computers and takes better advantage of memory and disk

space (Kivimäki & Rousku 1996:17,19-20, 22-24). It is also more expensive and needs more powerful hardware to work properly. If compared with Windows 95, Windows NT offers better performance, reliability and network management, including security issues. It can also be run on computers that have more than one processor, which Windows 95 is not suitable for (Kivimäki & Rousku 1996:19, 22-23).

Whether any of these features is needed from a corpus compiler's point of view is another thing altogether. The compilation of a corpus is mostly about processing of text, which generally does not demand huge processing power. Another downside to Windows NT is that not all ordinary Windows programs work in it; many corpus browsers may fall in this category.

When using a corpus, however, a Windows NT server could be used if there are plenty of users. The corpus could be physically situated on the server which also controls the access to it, keeping track of the users. Only the computer acting as the server needs Windows NT in it: the rest, the users' computers, can have Windows 95 in them (Kivimäki & Rousku 1996:20,24). In small networks, however, Windows NT client or simply Windows 95 could be used as a server.

Different versions of Windows have different requirements of the hardware. Windows 3.1x needs at least a 386SX processor or higher and 4 MB of memory. Windows 95 needs, theoretically, a 486 processor and 8 MB of memory; in practice it works very slowly on such a computer, and at least a Pentium processor and 16 MB of memory are recommended (Kivimäki & Rousku 1996:89-93). Windows NT needs at least 20 MB of memory to work at all, and 32 MB is recommended.

The biggest advantage of Windows in corpus work is that many computer users know how to use it and it is fairly easy to learn. Windows is also probably more readily available in most places than e.g. Unix. Many people would not want to get a new operating system just because of a corpus, and therefore, Windows is the natural choice for many users. It is also likely that people are familiar with other Windows programs that are needed to compile a corpus or process the data that the corpus provides, such as word processing programs. If you have Windows, and the means to scan and

proofread texts in it, you are likely to want to compile and process the corpus in it, too.

Windows is a good choice if the use of the corpus is considered. The researcher accessing the corpus is likely to want to save and study further the results of a corpus search. If one uses Windows most of the time, as many people do, it is definitely easiest to use the corpus in it, too.

Among the disadvantages of using Windows is that it is hard to say how long the corpus will be compatible with new versions of the operating system. Windows is getting updated all the time, the direction of the developments nowadays being toward Windows NT. There are also many people who predict that Windows will soon disappear altogether and give way to other operating systems, such as Linux. It is impossible to say what kind of an operating system people will be using in ten years and whether it will be at all compatible with Windows or not. Therefore, one cannot expect that a corpus browser for Windows 95 can be used after a number of years; indeed, the pace of development of computers is likely to make many programs old in less than a year.

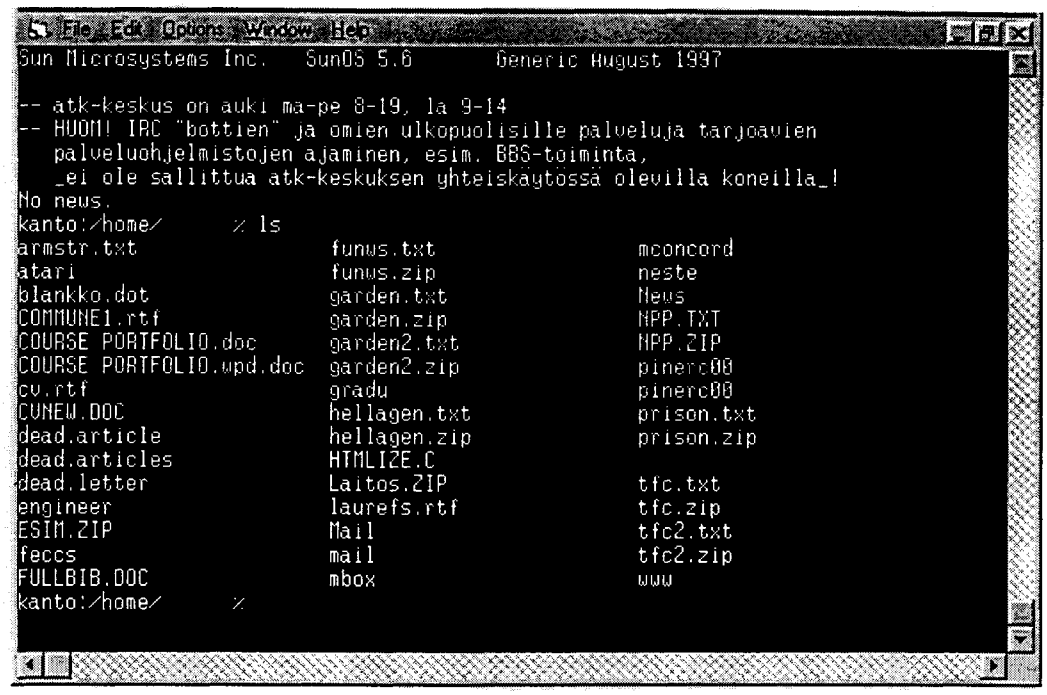
2.1.4 Unix

Unix began at AT&T's Bell Laboratories in 1969 (Lawler 1998:139). It is a multi-user, multi-tasking system, which is available for many different kinds of computers (e.g. Sun, Hewlett-Packard, MIPS, NeXT, DEC, IBM) (Lawler 1998:138, Dougherty & O'Reilly 1988:1). It is mostly used in large mainframe computers, rather than in personal computers. Multi-user means that many people can use the same computer at the same time; multi-tasking means that each user may have several different programs running at once (Lawler 1998:152). This practically means that the system gives milliseconds at a time to one user, switching between applications and users, constantly going through all the users and tasks at hand (Eloranta et. al. 1994:3, Byers 1985:2). A Unix system run in a huge central computer is far more powerful than any personal computer. On the other hand, there are versions of Unix that can be run on fairly small computers, too.

There are some distinct advantages of Unix from a corpus user's point of view. One of them is that the corpus can be situated physically in one place, in one computer, and several users can access it at once. The central computer takes care of the file-sharing and enables several people to read the same files at the same time, with no need to be aware of one another. Other client-server solutions can do the same, but Unix also provides exceptionally efficient means to control who, exactly, gets to use any programs or files.

All users need a login name and a password that gains them access to the computer. These, at least theoretically, prevent unauthorized people from getting to the files and protect the data from outsiders (Byers 1985:13). All the users also belong to a group, or several of groups. Files can be protected so that only the users who belong to a certain group can read or write to them. It might be, also, that several groups of users can read certain files, but only a certain group, or a certain user, can write to them (Eloranta et al. 1994:3, 84-85). All this makes the system more secure than many others. Even though other systems use password authorization, too, Unix is often considered to be the most reliable.

An ordinary, character-based Unix program is not dependent on the operating system of the user's personal computer, either. Simple terminal emulation programs are available for many kinds of computers and operating systems. A terminal emulation program provides the user with an interface that can be used to access the central computer. The actual program, such as a corpus concordancer, is run in the central computer. The terminal emulation program is only representing it on the screen of the user, in ASCII characters, which are not system-dependent. An example of such an arrangement is the Cobuild*Direct* corpus, which can be accessed by a telnet connection (or by a Web browser, nowadays) by anybody who has a user name and password for it (go to http://www.cobuild.collins.co.uk/direct_info.html for more information). Below is a picture of a terminal emulation program. No specific program is being run with it; there is simply some of the login messages and a directory listing visible on the screen.



Picture 3: Terminal emulation

Another important point is that Unix is not so prone to major changes as other systems. Though it began 30 years ago, it is still widely used. Unix is especially known as the favorite of computer nerds and other people who use computers very much. There are, in fact, jokes about Unix being a religion rather than an operating system. It is not likely to disappear for quite some time, because there are quite a lot of people who would not even consider using anything else.

The other side of the coin is that there are several slightly different versions of Unix. In fact, the whole system more or less consists of "building blocks" that can be combined in different ways. Unix is based on the idea that the user is provided with simple tools that each perform one single task, little individual programs, that can be "snapped together" for larger tasks (Lawler 1998:139-140). After the initial versions, Unix was given to users with the original source code so that anyone could modify it, resulting in many slightly different versions. Nowadays, there are mainly two versions of Unix: BSD (Berkeley Standard Distribution of Unix) which is free, and System V, which is AT&T's commercial version. There are significant differences between these

two (Lawler 1998:144). In addition, there are yet other versions and local differences.

The differences in Unix versions and configurations may not be relevant to the user, however, but do concern the programmers and the programs that can be run in a specific Unix system (Eloranta et. al 1994:3). In the long run, one can be fairly sure that Unix will be around; however, at any given point in time, one will come across a variety of them.

In all, its portability, its ability to run on many different kinds of hardware and its longevity make Unix particularly useful in case of a corpus that is going to be used for a number of years. As more and more powerful hardware becomes available, Unix can be installed to it and any programs and data can be easily moved to the new hardware (Dougherty & O'Reilly 1988:8). This in addition to the fact that Unix is a good choice if there are plenty of users who need to access the corpus from many different places.

There are also plenty of corpus software available for Unix. Many large, well-known corpus projects were made in Unix and many still are. In addition, the other programs that are needed in the compilation and use of a corpus, such as word processing tools, are readily available in Unix in any case.

Lawler (1998:164) also makes an interesting point mentioning that since the system has not changed very much since the 1970s, there are plenty of older books that are still useful. This can really be an advantage to someone who is trying to learn to use Unix: any library probably has plenty of books about it, and the information is still relevant. In many other operating systems this is not so; rather, a new manual is needed with each new version of the software. It may also be mentioned here that there are incredible amounts of information about Unix, and software for it, available through the World Wide Web and other networks. That, of course, can be a bit of a disadvantage, too; it may be hard to find relevant information among all the rest.

A downside of Unix is that if you do not have it, you are not likely to get it either. You are not likely to get a large Unix system just because of a corpus. It is far too expensive and demanding to get and maintain if there is no other demand but a corpus for it. There are also cheaper versions of Unix, such

questioned how much of an advantage that really is. Do people want to learn a new language, or do they want to learn to click with a mouse? Especially for newcomers to computers, Unix might still not be the ideal choice, no matter how interesting it might be from a linguistic point of view.

To sum up, Unix is a good choice for large corpora with many users, and corpora that need to be available for many years. There is also plenty of software available for it. However, it is not readily available everywhere, and it is likely that there will have to be a person whose sole task is to take care of it. Most larger universities and other institutions have Unix machines available, and a local computer center that takes care of them. In that case, it may be possible to arrange the storage of the corpus with them. Depending on the local arrangements, it may incur expense. If you plan to keep a corpus in Unix, you must find out about what is locally available and what the terms of usage are for the Unix system.

2.1.5 Other

There are also several other operating systems available, such as DOS, Macintosh OS for Macintosh, OS/2, and Linux. This section will briefly discuss them.

Several older computers still have **DOS** in them. There are also several corpus programs, such as MicroConcord, available for DOS. Although such programs may be old, they may be just as useful as any new ones.

If you have only older computers to use, and you are not willing or able to update to anything more recent just because of a small corpus, it is definitely worth considering to use DOS. Even though it is probably easier to scan and proofread texts in Windows, there is no reason why you could not use them in a DOS-based program in the end.

However, in case of larger corpora that one wants to use for years to come, it is worth considering how to store the data if it is compiled in an operating system that has already mostly disappeared from use. ASCII text can be moved fairly easily and processed further in any operating system, but if the corpus program uses its own file format (such as the database of TCE, which can be read only by TCE), it is not a good idea nowadays to choose a DOS-

based program. It would have to be updated to another file format anyways in a very short time. However, the same problem may come up with any other operating system as well.

Macintosh is a fairly popular computer among researchers in the humanities. The user interface is very user-friendly, and mostly it does not demand any knowledge about what actually is going on inside the computer. It is a rather good option for people who value the ease of use above all else. There are also corpus programs available for Macintosh, for example a concordancer called *Conc* (Antworth & Valentine 1998:194).

OS/2 is a rather rare operating system. It has a graphical user interface and it is known to be quite reliable. It works on personal computers and it is, in fact, considered by many people to be better that Windows (cf. Kivimäki & Rousku 1996:14). The problem that is most likely to come up with OS/2 is the lack of other users, and therefore reduced possibilities for sharing files and using the same software.

Linux, originally a version of Unix, is fast becoming a worthwhile opponent for Windows. It is frequently named as the operating system of the future, and is expected to keep on getting more and more popular. Nowadays, however, it is still mostly favored by computer experts rather than so-called ordinary users. It may become very popular in the future, though, and should not be discounted as an option.

The problem with all less common operating systems is simply that they limit the number of potential users of the corpus. It may also be very hard to find suitable corpus browsers for them. If the corpus is simply for the researcher's own use, it does not matter how rare the system is; however, with a big corpus that many people want to use in their own computers, the less common operating systems are more or less ruled out to begin with.

This chapter has dealt with the issue of selecting suitable hardware and operating system for compiling and using a corpus. The beginning of this chapter gave some general guidelines about choosing hardware and operating systems: the rest of the chapter introduced two of the most popular systems,

Windows and Unix, in more detail. The last section briefly dealt with other, less common operating systems.

The aim of this chapter was to introduce the systems and list their advantages and disadvantages rather than give any definite rules for choosing the hardware and software. What actually is the best option depends on many factors. What is already available, what the users can use, what can be bought, how much it costs, and what the users are willing to learn, to mention only some of the questions that need to be addressed. Often the corpus software that is going to be used dictates the environment; sometimes, a project starts from scratch, so that the programs can be tailored to suit any kinds of machines. In addition, the corpus may need to be moved to another environment later on, or new users wish to join, so that some alterations need to be made.

Note that very large and very small corpora have different needs. In case of a very small corpus it does not really matter at all where and how one chooses to store it; if it is for the researcher's personal use only, it can be stored in the way that seems the most convenient to the user. If it needs to be moved, it will not take much time. If it needs to be converted to a completely different format, the time and effort required is probably within reasonable limits. However, a large corpus will cause problems in such situations, and its storage has to be planned much more carefully. Of course, careful planning always cuts down the number of difficulties later on.

It may be that sooner or later, it all comes down to money. The wisest course may be to try to use what you have got already: do not buy anything new unless there is a pressing reason to do so. There are plenty of corpus programs available, and it is very likely that you will find something that suits your purposes as well as your hardware.

The day may be ahead, though, when the hardware and operating system are too old and the corpus needs to be moved to newer surroundings. It may also be that another researcher, on the other side of the world, wants to have a look at the corpus material. In that case, it is an advantage if the data is stored in a format that is operating system independent, and can be read by many different programs. Many corpus programs, indeed, save their data in plain text format (or can convert it to plain text) so that it should, theoretically,

be easy to move from an environment to another. There may be, however, problems even with plain text files. The following chapter examines character sets, problems that may come up with plain text, and character set problems in general.

2.2 Character Sets and related issues

This chapter is about character sets, plain text formats, fonts, and the problems that they may cause in the compilation and use of a corpus. The compiler and user of a corpus may, at one time or another, run into problems with different character sets. The researcher who studies a language that uses a non-Roman script will surely have to consider how to make the characters correctly visible on screen: in many cases, anyone who deals with language other than English may encounter problems since some special characters are missing. The compiler of a corpus who has to move the texts from an operating system to another, or the user who receives the texts by e-mail from a friend with a totally different kind of a computer, are likely to get familiar with character set problems.

There are several different things to be considered here. First of all, all computers have a basic **character set**, and they can save data in a so-called **plain text format**, which has only the text and no other formatting, such as italics. In addition, many computers and programs offer the choice of several **fonts**, which can be used to affect the looks of the characters that are printed on screen or paper. These are two different things, and should be kept apart.

A plain text file is a file that includes only the text itself: it does not contain any other information, such as word processors' formatting parametres. A plain text file is not specific to any particular program, either. Many other formats, such as those that word processors use, are meant for a certain program: for example, a file saved in Word 97 format can be read only by Words 97, and possibly by WordPerfect. However, a corpus concordancer could not read a Word 97 file. Some other type of computer except an IBM-PC with Windows 95/98 could not read it either, unless Word 97 were available to it. In order to use a Word 97 file, you need Word 97 and a computer that runs it. A Word 97 file is a binary file that does not even look like text, if accessed with a program that cannot read it properly.

A plain text file, however, is not dependent on specific programs. It can be read by any program that reads text files. Any word processor can read plain text. In addition to this, practically all kinds of computers can read plain text. This is the reason why many corpus programs use a plain text format: it is

available to everyone, it is easy to save files to plain text format, and almost any program can read them. Plain text is indeed the file format that seems to be most common in storing corpus data, and therefore it is important to any corpus compiler and user to know about it.

A character set used by many computers nowadays is the **ASCII (American Standard Code for Information Interchange) set** (Deitel & Deitel 1994: 338). Plain text files are, in fact, often referred to as **ASCII text** files. The ASCII code set gives a numeric code for each character. For example, the code for a capital *A* is 65. This is the how the computer sees a stretch of plain text: it is a sequence of numbers, each number denoting one character. The full ASCII character set is in the following table:

Table 1: ASCII character set

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	I	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

The digits at the left of the table are the left digits of the decimal equivalent (0-127) of the character code, and the digits at the top of the table are the right digits of the character code. For example, the character code for 'F' is 70, and the character code for '&' is 38.

(Deitel & Deitel 1994:891)

It may be noted that the code set also includes codes for "characters" that are not important to the ordinary user (codes 0-32). They are meaningful to the computer (e.g. code 13 *carriage return* and 27 *escape*) but most of them do not concern the user who deals with normal alphanumeric data.

As can be seen in the above table, the ASCII code set includes codes from 0 to 127. From a more technical point of view, this means that it uses only the first 7 bits of a byte (a byte consists of 8 bits). In order to include more characters, such as *ä* and *ö*, there are 8-bit character sets which can utilize the codes 128-255 and thus include twice as much characters as the normal ASCII set.

However, the problem here is that these additional characters do not have the same code in different operating systems (Lawler 1998:147, Hirvonen 1994:118-119). Lawler (1998:147-148) uses the word Low ASCII for codes 0-127 and High ASCII for the codes 128-255. He notes that Low ASCII is completely standard, but High ASCII is different in different types of machines. He points out, for example, that although DOS/Windows and Macintosh text files have many of the same additional characters, these characters have different codes. In practice this means that even plain text files are not completely compatible between different types of computers, because some characters may be represented incorrectly.

Another problem between different types of machines and operating systems is that they represent the end of a line in different ways. ASCII text is often read from file one line at a time. The end of the line is signalled by an end-of-line-character, which is stored in different ways in different operating systems. For example, Unix uses ASCII character 10 (*linefeed*), whereas Macintosh uses ASCII character 13 (*carriage return*) for end-of-line. DOS, however, expects to find both of these characters to signal a new line of text (Hirvonen 1995:83). Note that these kinds of "characters" relate to the fact that the ASCII system is quite old: *linefeed* and *carriage return* refer to an old type of a printer, which first moves the paper up one line (*linefeed*) and then returns the writing head to the beginning of the line (*carriage return*), like a typewriter. In practice this means that when texts are transferred from one

operating system to another without any file format conversion, it may be that all the line breaks have dissappeared and the whole text is just one huge line.

Connected to this, yet another problem that a compiler or a user of a corpus may run into is that different programs have different ideas about how long a line should be. How likely this is to cause problems depends entirely on the kinds of programs that are used. With commercial programs such problems are unlikely; however, homemade programs may make assumptions that cause problems.

One problem that came up when compiling the FECCS corpus at the University of Jyväskylä was that one of the programs that added certain tags to the texts expected that lines should be only up to 80 characters long. The ASCII format that the texts were stored in, however, allowed longer lines. The tagger program simply did not read all of the lines; it read only the first 80 characters of each, and then jumped to the next one, thus cutting off the ends of many lines.

Therefore, if a compiler or user of a corpus of ASCII text encounters weird problems, such as seemingly endless lines when there should be only short ones, or sentences that end abruptly, it is worth checking what kind of ASCII text one has got, after all. The only solution here is to try to save the text again in a slightly different format (most word processors offer more than one kind of ASCII text) and see what happens.

It should be mentioned here that it is easier to revert back to the original from some plain text formats than from others. For example, if you have used a format that for one reason or another inserts an end-of-line-character at the end of every line on the screen (some of them do), any programs that are used later interpret each line as a separate sentence and paragraph. However, getting rid of the end-of-line-characters may not be easy, because the computer is not likely to know which ones are not the extra ones. It is a good idea always to save the text in more that one format until you are certain which one works best.

It is also possible to keep the texts in a specific word processor's format as long as they are still being proofread and preprocessed, and only convert them to ASCII when they are ready to go to the corpus browser. This

procedure has the advantage that it is usually easy to convert a word processor's file to any type of plain text. The disadvantage is that the user might inadvertently use characters or other codes that are not available in ASCII, so that the file gets more or less messed up when it is converted to plain text.

In addition, not only the differences between different ASCII systems may cause problems, but also the limitations of any type of ASCII. As Lawler (1998:147) notes, it is more or less impossible to represent other but western, Roman script languages with it. Even many European languages encounter problems with ASCII, if they require many extra characters, such as *ä, ö, æ, ñ,* or others that are missing in Low ASCII.

One problem that may also come up regards alphabetizing. For the user of the corpus this is a rather important point, since concordances are usually retrieved and sorted in alphabetical order. The most simple type of alphabetizing that the computer can do is based on the fact that the characters happen to be in alphabetical order in the ASCII chart. The computer knows that, for example, a word starting with a character that has the code 65 (*A*) comes before the character that has the code 66 (*B*), simply because 65 is smaller than 66. Lawler (1998:148) also points out that the difference of codes between an upper-case character and the corresponding lower-case character is always exactly 32, so it is easy for the computer to see their relationship. However, any extra characters, like *ä* and *ö*, have codes that are not in any logical relationship to the rest of the alphabet, or to each other. The computer does not know where they belong, unless the program can specifically deal with them. To make matters even more complicated, in some languages alphabetizing is not done as in English. For example, in Spanish, *cuatro* should be before *chapotean*, because there are some character combinations in Spanish (*ch, ll, rr*) which are alphabetized as if they were single letters (Hockey 1998:119).

Of course, many programs can deal with the alphabetization correctly. For example, if a concordancer is meant to be used with Finnish, it certainly knows where to put *ä, ö* and *å*. However, if you tried to alphabetize Spanish

with it, it probably would not succeed at all. The corpus browser that is used should be able to deal with the language that is processed in it.

There are a couple of possible solutions to this type of character set problems that involve characters that are missing from standard ASCII. First of all, in Windows and Macintosh it is possible to use different **fonts** (Simons 1998:12). For instance, you might have a normal Roman script computer, but install a Russian Cyrillic font to it, so that you could also view Russian texts. The problem with this, however, is that you also need corpus software that supports different fonts. Most Windows and Macintosh programs probably do, so that this is not a real problem there. In DOS and Unix this may prove practically impossible, however, since they have a fixed set of characters.

There is also a character set called **Unicode**, which includes 38,885 characters from all the major writing systems of the world (Simons 1998:14). It is, however, meant for the exchange of data and is not equally well suited for linguistic research. For example, it is not aware of what language it is used to encode, and therefore cannot consider language-dependent factors (Simons 1998:14). Unicode is also a rather new system yet. However, with time it may become widely used for interchange of data. Exchange of resources is indeed one more of the reasons why character sets are important for the compiler and user of a corpus.

A corpus user might, sooner or later, want to give the data to another researcher who lives in another country and uses a different type of computer. If one compiles and uses a corpus in just one computer, in only one operating system, there are, in fact, no real problems with the character sets: one should simply use the one that looks and works fine (Sperberg-McQueen & Burnard 1994:81-82). However, if one needs to give the database of texts to other people, there may be such problems as were referred to earlier: some characters may be represented incorrectly in another environment. The same problem may come up locally, too, when the files are moved to another type of computer and operating system.

There is a problem familiar to all those who write e-mail; sometimes, *ä* and *ö* have been changed into other characters such as *{* and | , sometimes they are replaced by weird sequences of codes, sometimes with other letters.

This can be either because the the two operating systems or programs (the one the e-mail was sent from and the one that received it) use different codes for *ä* and *ö*, or the receiving system is a 7-bit system which simply ignores the 8th bit and misinterpretes the code. As McEnery & Wilson (1996:33) note, mainframe computers in English-speaking countries tend to use 7-bit character sets. This means that such computers are capable of dealing with only the first 128 codes in the ASCII set, and cannot represent languages other than English correctly.

Therefore, sending corpus texts as an ordinary e-mail is not a particularly good idea. However, most modern e-mail programs have the option of sending an attachment with the e-mail message. An attachment should arrive to the recipient in exactly the same form as it left, providing, of course, that the recipient can receive e-mail attachments. However, it does not remove the problem that the sender and recipient might have quite different types of computers and character sets nevertheless, so that even though the e-mailing of the material does not mess it up, some other stage in the process might.

Another way to deal with this is to send the file in Unicode, or turn it into text that has only ASCII characters (Dry & Aristar 1998:34). Unicode is not very common yet, but the latter option, although a bit arduous, might prove useful. The TEI Guidelines, which will be dealt with in chapter 3.4, give a few suggestions for encoding texts so that no characters are lost.

Sperberg-McQueen & Burnard (1994:86) observe that the characters least susceptible to get mixed up are those that belong to the ISO 646 subset. It includes the following characters:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
" % & ' ( ) * + , - . / : ; < = > ? _
```

(Sperberg-McQueen & Burnard 1994:86)

In order to make additional characters survive different operating systems and interchange through computer networks, only one procedure is guaranteed to work: replacing all special characters by what are called **SGML**

entity references (Sperberg-McQueen & Burnard, 1994:82, 86). (Chapter 3.4 will deal with SGML in more detail.)

Entity references can be used to replace otherwise problematic characters. They are strings of characters that are preceded by an ampersand and followed by a semicolon (Sperberg-McQueen & Burnard, 1994:82). For example, the entity reference for *ä* is *ä*. Thus, a Finnish word like *änkyttävä* could be changed into *änkyttävä*. Since the entity reference codes themselves consist only of characters that are safe to transfer, using them ensures that the text survives intact.

The problem here, from the point of view of corpora, is of course that the words with entity references in them are more or less impossible to read as such. The corpus browser should be able to decode them back to intelligible characters. However, how many programs can do that, is another matter altogether. The browser should have been designed to read such texts in the first place. This matter can be generally referred to as annotation awareness and it will be shortly addressed in chapter 2.4.

One possible solution is to replace problematic characters when needed, for example when the texts are sent to other researchers. The others could then, with suitable instructions, convert them back to what they were supposed to look like. How time-consuming a task this is to the user depends on how many texts there are, and whether or not there is a program that is able to do it automatically to all the texts.

It is relatively easy to find and replace characters in any word processing program, and it is possible to program a macro that changes all entity references back to normal characters with a single click. Theoretically, it is not very hard to write a separate program that does the whole thing automatically. However, the researchers may not be able to do this. In case of a large corpus with many potential users it should be considered as an option, though, to make separate programs for such tasks.

In this chapter I have dealt with the problems that may come up with character sets and plain text formats. Many corpus browsers, rather than using a format of their own, use texts in plain text format. This has the advantage that plain

text is available on all kinds of computers, and it is not tied to any specific programs. The disadvantage, however, is that the character sets that plain text uses tend to be a bit different in different kinds of computers, and they are not fully standardized. This causes problems when the text files are moved to a different type of computer, operating system, or another country, where the character set is slightly different. Nevertheless, these problems can be dealt with, for example by replacing all the problem-causing characters with SGML entity references during transfer.

Whereas this chapter shortly explored the problems that may come up with moving the files, the following chapter will consider the other side of the matter: where and how to store the information locally.

2.3 How to store information & Backup copies

This chapter briefly deals with the matter of storing the information. The corpus has to be physically stored somewhere, and backup copies need to be taken at regular intervals. This chapter lists some of the options for storing the data.

The options for storing the information are not, in fact, very different for different kinds of computers. This chapter will not, however, explore issues related to Unix or other network solutions. If a corpus is to be kept in a Unix system, it is best to leave the practical hardware issues to the local computer center or whoever is responsible for the system. This chapter will, instead, look at the different storage options for personal computers.

What is the best medium for storing the information depends on the size and uses of the corpus. For some purposes, it is enough to have a few computers where the corpus permanently resides; for some other purposes, like teaching, it may be necessary to use the corpus in many computers and a number of different places. A small corpus can be moved easily from a place to another, even on floppy disks; a large corpus, however, may prove to need so much disk space that it cannot be moved effortlessly. The following sections will give a few examples of means of storing information, their pros and cons, and for what purposes they are good.

2.3.1 Hard disk vs. floppy disks

The hard disk of a personal computer is the most natural place where to keep information while compiling a corpus. Each personal computer nowadays has a hard disk; the size of it may vary, depending on the age of the computer, somewhere between 500MB and 10GB. The hard disk is definitely a better option for storing data than floppy disks. Many people (students, especially) seem to prefer floppy disks for storing their files, but one should avoid floppies if possible. It is true that if one cannot work on the same computer all the time, it is necessary to carry floppies around; however, if there is a computer that can be used for a given task continuously, the files should definitely be kept there.

The main reason for this is simply that floppies are rather unreliable. A hard disk may fail, too, but that is not nearly as likely as a floppy disk refusing to work. The hard disk is much better protected from outside interference such as dust or magnetism, and it is meant to be used more or less continuously. Of course, a hard disk is also much larger and faster than floppies, which are more or less relics from a previous age, nowadays.

In case of a small corpus, however, floppies can be used for taking backup copies and moving the files from one computer to another. If the corpus data amounts to only a few megabytes, or less, it is not too much trouble to keep copying it onto floppies. The advantage with floppies is that almost all personal computers nowadays still have a floppy disk drive; therefore, small amounts of data are easily moved on them.

There is the problem that in recent years floppies have become increasingly insufficient for storing data because of their small capacity. An ordinary high density floppy disk takes only 1.44MB of data, and that is not very much. The problem is that there has been no new hardware that would have replaced the floppy drive as the "standard" disk drive on all personal computers. Several kinds of higher capacity disk drives are available, but none of them has gained a position over the others. The following section will say a few words about them.

2.3.2 Other disk drives

Since floppy disks are not large enough any more for many users' purposes, other types of disk drives have emerged. Because the development in this area is fast, as in the field of computers in general, it would not be useful to present all the different kinds of drives here. Any recommendations are bound to change rather quickly. However, to give a general idea of the new drives that are available, it is good to introduce one of them as an example. One of the most popular of the higher-capacity disk drives is the **Iomega Zip** drive.

The Iomega Zip drive is a disk drive that reads and writes 100MB disks specifically made for it. In early 1999, Iomega also presented its new Zip 250 drive, which reads 250MB disks in addition to the 100MB disks (www.iomega.com). Note that despite its name, it should not be confused with

zip-compressed files. The Zip drives are usually external, although some computer manufacturers have made computer sets that have a build-in Zip drive in them. The disks look very much like ordinary floppy disks, except that they are thicker, heavier and much more expensive. Although the Zip drive itself is not very expensive (around 1200 FIM at the time of writing) the disks cost about 100 FIM each.

One advantage of the Zip drive is that both the drive itself and the disks are easily portable. At the same time, they provide quite a large storage capacity if compared to floppy disks. Therefore, they provide a good option for both backup copies and for moving the data from a place to another.

The Zip drive has some disadvantages, too. The most notable of these is that although it is very popular, it is no standard: they do not exist everywhere. Therefore, in addition to taking the disks with you to a different town, you may end up dragging the drive with you, too. It is rather easy to install to another computer, but it still takes time. The drive may also seem rather slow, if compared to a hard disk. There have also been suggestions that it may not be as reliable as desired.

The Zip drive is a good option for backup-copies of a moderate-sized corpus that is changed sometimes or still incomplete, but does not need to be moved very often, nor installed to a large number of other computers. If the corpus is very large, however, and needs to be constantly used in different machines, a CD-Rom copy may be a better option yet.

2.3.3 CD-Rom

Most personal computers nowadays have a CD-Rom drive; new computers all do. A CD-Rom can take up to 650MB of data. There are lots of differences regarding the speed of CD-Rom drives, but even the slowest models are rather fast, if compared to other storage media.

A normal CD-Rom drive only reads CDs. However, there are also so-called CD-R drives (short for Compact Disk Recordable). These drives also write CD-Roms. They use CD-R disks that are specifically designed to be written on. Such disks, however, can be read by any CD-Rom drive, with the exception of some older models that may not accept them.

The limitations of CD-R include that the data on the disk cannot be removed. Although the disk does not have to be written all at once (information can be added to it later), the data on it can never be removed. In other words, CD-R disks cannot be reused time after time, as the information on it stays there permanently.

CD-R drives cost about 1500-4000 FIM at the time of writing. The disks, however, are rather cheap: depending of what the disk is made of, the price is somewhere between 10-20 FIM each.

The advantages of CD-Roms include that they take in a huge amount of data, and that they can be easily used pretty much anywhere. The downside, however, is that the information on them usually cannot be changed. Therefore, they are suitable for storing a finished corpus that is not under any construction anymore. CD-Roms cannot be very strongly recommended for continuous backup copying or any other recurrent saving of incomplete data. However, since the CD-Rs are rather cheap, they can be used for occasional backup copies and other storage, if the price of constantly buying new CDs is not too high. CD-Roms are excellent for the storage of a finished corpus, though. Researchers and students can take the CDs home and use them on their own machines, and any kind of teaching is easier to arrange, since the data does not have to be installed on a new computer but can be used directly from the CD-Rom.

However, there are also so-called CD-RW (Compact Disk ReWritable) drives. These drives use specifically designed CD-Roms (CD-RWs) that can be written multiple times, that is, the information on them can be removed and the disk can be used for storing other data. These disks can be read only by a CD-RW drive. Many recording CD-drives nowadays can use both CD-R and CD-RW disks. The CD-RW disks, however, are still rather expensive: they cost about 75 FIM each. CD-RW drives are still rather new.

The word "backup copy" keeps propping up in the preceding sections every now and then. The following section gives some general advice on the matter of taking backup copies.

2.3.4 Backup copies

You can never have too many backup copies. There are all kinds of problems that can result into the loss of data, like hard-disk failures. Backup copies should be taken regularly and kept in a safe place.

It is hard to say how often backup copies should be taken. It depends on the project, how much data there are and how often new data comes in. Any new data, however, should be copied to a safe place as soon as a certain stage with it has been achieved – for example, when a text has been fully scanned, or proofread, or tagged.

For an ongoing project, it is useful to take backup copies of new data as often as possible, every day or a few times a week. Such backup copies can be taken on floppy disks, since new pieces of data are usually not very large. A more thorough backup copying (with a Zip drive, for example) can take place every week or once in two weeks. When a certain stage has been reached (e.g. all texts have been scanned, or a whole category or subcorpus is finished), the data can be written on a CD-Rom. This is only a suggestion; it is in each project's own interests to define how often backup copies need to be taken and how much damage a possible loss of data will cause.

Although the beginning of this chapter specified that Unix and other network solutions are not considered here, a few words are still necessary. It may be a tempting idea to put the data to a central computer or a file server, so that people could access it each from their own computer. However, there are a few things that need to be taken into account here.

The permissions to use the texts do not automatically include the right to put them into some kind of a network. Putting the texts to a public network, such as the Internet, is definitely illegal without proper permissions for it, even if the material is protected by passwords. Before making any such plans, one should find out how far the permissions of use actually extend.

Secondly, it may be that the software that is used with the corpus does not work if the data is not on the local computer itself. One should check whether the program can be used over the type of network that is planned before taking any other steps towards it. One also needs to find out about the

licencing agreement of the corpus software: the permissions for its use may limit any networking plans rather severely.

Of course, whenever information is put into any kind of computer network, the question of security arises. It is an issue that is probably practically handled by the local computer center, or whoever is responsible for taking care of the network. One should, however, be aware that hackers and other unexpected visitors may take an interest. The corpus should be protected well enough so that only people who have a permission to use it are able to access it.

Note that wherever you keep the files, it is worthwhile keeping track of what is kept and where, especially during the compilation of the corpus when there may be several versions of the same texts around. On a single computer, making a directory structure that is clear helps to keep certain types of files in a certain place. In addition, the files should be given names that tell instantly what version of the text it is, e.g. *name_s.txt* for a scanned text and *name_p.txt* for the proofread version.

If the files are kept on more than one computer, the situation is more complicated because you will have to know what is on which computer. Extra copies of the same file tend to accidentally remain on different computers, and it may also be the case that one computer has a newer version of the same file than another. If these types of file management issues are not taken care of properly, it may be that soon it is very hard to know which files are the newest ones and which ones should be deleted.

This chapter has briefly introduced different types of hardware for the storing of the data, and given some general advice on the subject of backup copies and file management. The following chapter, in turn, will explore the matter of software.

2.4 Programs

One decision to be made is about the acquisition of the programs that will be used to compile and use the corpus. Basically, there are a few alternatives: get a program that was developed for another corpus project, use a commercial program, make the program, have someone else make it, or find a free program that can be used. There are both pros and cons to each of these alternatives. The choice depends on many factors: what kinds of programs are already available, what kind of a program you need, how much money you are willing to use, what kind of a corpus you have, and so on.

Before selecting a specific program, you need to know what exactly you want to do with it. This highly depends on the kind of research that you are going to do, so it is not possible to give any exact advice about it here. There are, however, a few general requirements that can be mentioned.

First of all, the program should have certain basic functions, such as concordancing and searching for collocations, production of frequency lists and a flexible query syntax (McEnery & Rayson 1997:203). The last point means that the user should be able to use boolean operators, such as AND and NOT, and wildcard characters such as the asterisk (*) to define searches. The program should also allow for user-defined subcorpora and be able to give details of the text where a retrieval instance came from (McEnery & Wilson 1997:203).

Part 3 of the present paper will deal with the matter of **annotation**. Many corpora in the world are, in one way or the other, annotated. A very common type of annotation would be, for example, part-of-speech annotation, which means that each word in the corpus is given a code that tells which word class it belongs to. When searching such a corpus, the annotation is of limited use if the program does not recognize the annotations as what they are but sees them as parts of corpus text. McEnery & Rayson (1997:203) note that **annotation awareness**, i.e. the program's ability to recognize the annotations, is a function to be expected in any software. Such a program should be able to hide the annotations and show only the corpus text to the user, and it should also enable the researcher to search on annotations. For instance, the user could

search for all the cases when *will* is used as an auxiliary verb, and get only those instances, not all the other *wills* in the corpus.

As was pointed out in chapter 2.1, choosing the program is tied to the matter of choosing the hardware. Very few corpus programs are available for many different kinds of computers and operating systems (McEnery & Rayson 1997:203). For example, if the potential users of the corpus use both Windows and Macintosh, the ideal would be to find a corpus program that is available for both systems. On the other hand, as McEnery & Rayson (1997:203) point out, in the future many corpora may be accessible through the World Wide Web, which is independent of the type of machine that is used to access it. In other words, it is possible to put the corpus on a WWW server so that it can be accessed from any type of a computer with only a suitable Web browser.

There are a few general problems that may come up with any kind of software that is going to be used for linguistic research, not only corpus software. As Antworth & Valentine (1998:172) mention, there is not much linguistic software available for different kinds of tasks, and in some cases, the researcher may have to use general-purpose software, such as word processing or database programs. This kind of software, however, often does not suit the linguist's purposes very well: for example, there may be no support for dealing with more than one language simultaneously, or traditional database data fields cannot deal with linguistic units well enough (Antworth & Valentine 1998:172). Related to this, Simons (1998:10) points out that there is often a semantic gap between the linguist and computer programs. Programs tend to use computationally convenient objects, such as "files", "lines", "characters", "records" or "fields", and it may be hard for the linguist to try to fit linguistic objects, such as grammatical categories, into these.

Simons (1998:10) also mentions a friendliness gap between the software and the user. It means that some programs have one-of-a-kind user interface which is hard to learn: it takes time to learn it, and it can be forgotten easily. Another program might work in a completely different way, so that the user would have to learn it all from the beginning. This is especially typical of Unix software, while Windows and Macintosh programs are at least to some

extent standardized so that the same functions can be found in roughly the same places in any program.

The following sections will briefly describe the pros and cons of different ways of acquiring a program. Rogers (1998:85-87) lists common problems that may come up when acquiring software: that programs are available but they do not do what is required; that a program is under development, but never seems to get published; that the program is filled with programming errors; or that there is no money or other resources available for developing or buying suitable hardware and software. These are all interesting points to keep in mind, and the following sections will explain how they, in addition to other problems, relate to different types of programs.

2.4.1 A program that another project made for their corpus

There may be other corpus projects that have developed tools for their corpus and would be willing to give their programs freely or for a low cost. Although this option might otherwise seem very tempting, there is one problem with it. Programs developed for a specific project are usually made very specifically for their research needs and for their type of a corpus: they do not necessarily suite any other types of research. As Kahrel et al (1997:232) note, programs made for one group of researchers can rarely be re-used by other groups. This situation may be eased in the future, if annotation and other practices of corpus compilation can be standardized to a larger extent than they have been so far. The matter of standards will be explored in more detail in chapter 3.3.

2.4.2 Commercial programs

The biggest advantage with commercial programs, such as MicroConcord or WordSmith Tools, is their reliability. They have been tested not only by the manufacturer, but also by all the people who have used them already. Most bugs in them are likely to be known and fixed. The programs have also been made by professional programmers who, at least theoretically, should be able to make the program work logically from the users' point of view. In other words, a Windows-based corpus browser by professional programmers should

work as Windows programs always do, with no strange or illogical functions that homemade programs often suffer from. This is especially important to the inexperienced user of computers, for whom it is much easier to do things the customary way rather than work out never-before-seen procedures.

A commercial program should also come with a manual that not only explains how to use the program, but also assists with problems that may come up with either installation or use. There is often also a phone number or e-mail address that you can use to contact the manufacturer and ask for more advice. In addition, there are likely to be plenty of other, experienced users who can help with problems.

In other words, with a commercial program there is likely to be a lot of help available, and not many problems concerning programming errors and such. Once a program has been bought, it is also likely that the manufacturer offers updates to it regularly, such as new versions and patches that fix programming errors.

The downside to commercial programs is that they do what they were made for, and it is not necessarily what a researcher wants from them. If a researcher or a compiler of a corpus needs something very specific, such as a program that can align, say, English, Finnish, and Russian texts, it may be hard to find a finished product that can do it properly. Commercial, widespread programs are rarely made to do anything very specific; rather they offer procedures that can be applied to several different kinds of languages, such as simple concordancing.

2.4.3 Homemade programs

Another alternative is to make the program from the beginning within the project that compiles and uses the corpus. The immediate problem here, of course, is finding someone who knows how to program. Providing, however, that some such person is available within the project, this is an alternative worth considering.

The advantage with a program that is made for a specific corpus is that it will, at least theoretically, do exactly what is needed. Extra features can be added along the way, for as long as the programmer stays in the project. The

program can be developed according to the users' needs. This can be a huge advantage over commercial programs and makes different types of research possible. Biber et al. (1998:255-256) make a list of advantages that can be gained by writing your own programs: in addition to mentioning many of the previous points, they also note that self-made programs can be quicker and more accurate than commercial programs, and put no limits to the size of the corpus if the programmer so wishes. They also point out that in addition to allowing the type of searches that no commercial tools provide, the output of the program can be tailored specifically to the researchers' needs.

Also, if there are any problems with the program, help is near; the programmer, if anybody, should be able to help people with its use. It is much easier to ask someone next door than to try to get a commercial manufacturer's phone support line to answer at rush hours, and try to make them understand what exactly is the problem.

The problem with homemade programs is that there are no guarantees to their quality. The end result may be filled with programming errors and the workings of the program may be somewhat peculiar from the users' point of view. In the worst case, the program may be actually working incorrectly for a long time without anybody noticing, thus producing incorrect results. Sometimes, it may be that after a long time of unsuccessful programming the programmer finally has to give up, admitting that the program logic is wrong and it does not work as it was supposed to.

The person who did the programming may also leave the project at a rather problematic time. Not only will there be no-one any more to answer instantly the many questions that the users still have, but there may not be any more updates available to the program. If the programming was poorly documented in the first place, the development of the program may be at a dead end. If the documentation was done badly or not at all, it may be more or less impossible for someone else to try to find out how the program actually works. In that case it will be very hard for someone else to come and try to develop the program any further, so that the project may be forced to start using a new program altogether, if some new functions are needed.

This is not one of the cheapest options, either. The programmer has to be paid just like anyone else, and if he or she is hired only to program the corpus, it may become a bit expensive in the long run. Of course, theoretically, there is the option that one of the linguists already in the project learns to program, and Biber et al. (1998:256), in fact, heartily recommend it. Their attitude, however, seems a bit too optimistic. As was pointed out before in chapter 2.1, not many people in the humanities are very interested in computers as such, but rather use them only as tools to perform certain tasks that could not be done otherwise. Biber et al. (1998:256) are correct in the sense that corpus searches are mostly about manipulation of strings of characters, the programming of which is not a very difficult task if compared to some other programming tasks; however, if you feel like the kind of person who cannot tell one end of the computer from the other, it is best to let someone else take care of the programming.

If you feel like you are up to it, though, it is an idea worth considering to learn to program. Biber et al. (1998:256), in fact, recommend that one should take "a course in programming for linguistic analysis". If you happen to live in a place where such a course is available, it can indeed be recommended. Many corpus compilers, however, probably work in universities with rather more limited resources with respect to what kinds of programming courses are available, and have to do with somewhat less specified instruction. It must be noticed, though, that in any case learning necessary programming skills is going to take quite a lot of time, and may not be within the resources of the corpus project.

2.4.4 Program made for the project by people outside the project

The third alternative is to hire programmers from the outside. In practice, this means having a software company make the program.

If the design and programming of the new piece of software proves successful, this alternative may produce a program that combines the advantages of both the previous alternatives. The end result will be a professionally made, fully working program that does exactly what is needed. However, the downsides to this alternative are also many.

The most immediate downside is the price. However simple the resulting program is to be, it is bound to be expensive. A cheaper way to get a program would be to have it made by university students studying computer or information sciences, who might do it as a part of their studies or to earn a bit of extra money. A student project, of course, is a student project; there are no guarantees that it will succeed.

The same problem may come up with software companies, as well. There is nothing to guarantee that anything will come out of it, after all. And even if it does, will the program be exactly what was needed? The needs of linguists are not necessarily clear to the software manufacturers, and from the linguists' point of view, the program at hand in the end may not be quite what was wanted.

Continuity is yet another problem here. After the first version of the program has been paid and received, what about updates? What about error fixes, or new features, let alone new versions? How much money is that going to take? This option for the acquisition of corpus software should not really be considered by any other than fairly large projects with plenty of money at their disposal.

2.4.5 Freeware and shareware

Yet another alternative is to get a so-called **freeware** program. Freeware is software that can be freely distributed and used with no payments. Freeware programs are usually quite simple; when they get more complex, the makers often start to charge for them. **Shareware** programs are also freely available for testing, but if you start using them regularly, you are expected to pay for them. No-one controls whether you actually pay or not; however, there is a moral obligation to do so. Often, when you pay for a shareware program, you get some additions to it and the programmers will notify you of any new updates.

Shareware programs are usually much cheaper than commercial programs. Barnbrook (1996:27) notes that there are lots of differences in the conditions of use of freely available software. Some of them are completely free, or cost close to nothing, whereas others may cost almost as much as some

commercial programs. Both freeware and shareware programs are available through the Internet, and come with some kind of documentation and conditions of use.

Barnbrook (1996:27-28) notes that there are enourmous differences to the quality of both the software and its documentation in the case of shareware. He also notes that there may be problems with the installation. However, if you only need a very simply tagger, or concordancer, there might be proper freeware or shareware available. No-one can say how reliable such programs are, and certainly their use is on one's own responsibility. They might not do exactly what you need, but maybe just enough.

To end this chapter, it might be useful here to have a few words about **reusability**. The lack of compatibility of plenty of different types of corpus software has lead to an increasing need to have programs that would be suitable for many different types of corpora, and that could be used over and over again by different projects. In the future, it might be that instead of individual, incompatible programs there will be small pieces of software that can be put together and used as a one big program for a given task. McEnery & Rayson (1997:208) use the metaphor "software Lego" to describe such pieces of programming. This kind of thinking, of course, is nothing new: as was mentioned in section 2.1.4, the "snapping together" of tools was one of the basic ideas of Unix from the beginning. However, in the case of corpora, standardization of compilation and annotation practices is still required to make such programs possible.

This chapter has briefly explained the advantages and disadvantages of different ways of getting suitable software. Sometimes it may be the case that a project is willing to give out the program that they have used. It may not, however, suit other projects very well. The same problem may occur with commercial programs: although they are reliable and relatively bug-free, they are not tailored to any project's specific needs. A home-made program will indeed do what is wanted, but it may not be very reliable, and may be of limited usefulness to others. A program made for a project by outsiders might produce excellent results, but the price may become a problem. None of these

options is bound to be the perfect solution, but all of them have their advantages. It is also worthwhile to search the Internet for suitable freeware and shareware: although not as refined as commercial programs, nor tailored for your specific needs, they might indeed do exactly what is required. In fact, a useful list of links to corpus software (both free and commercial) can be found at the W3 Corpora site at http://clwww.essex.ac.uk/w3c/corpus_ling /content/software.html. Whatever the needs of the corpus project are, some kind of a compromise can be found to get software that is not too expensive, but does what is required.

2.5 Summary

In this part of the present paper I have explored computer matters that are relevant for the compilation and use of corpora. Since corpora are nowadays invariably kept in computers, there are decisions to be made about both the hardware and the software. There are also many sources of problems that one has to be aware of, such as different character sets. One also has to consider how to store the data and how to move it from one computer to another as easily as possible. These are the kinds of questions that this part of the present paper has addressed.

In chapter 2.1, I discussed hardware and operating systems. The chapter gave general advice on what kinds of factors should be considered in the selection of computers and operating systems. Although a few operating systems were introduced in more detail, it should be remembered that both computer hardware and software develop very fast and it is the general principles that should be remembered rather than the details. It should also be remembered that the decisions should be made bearing in mind what kind of a corpus is going to used, and what kind of a system is the best for it and its users.

Chapter 2.2 told about character sets and problems that may come up with them. All computers are able to save texts in some kind of a plain text format; in addition, many programs can use different fonts, which enable the use of special characters that are not part of the basic character set of the computer. Many character set problems also have to do with moving the data. Whenever a plain text file is moved from a computer to another, on disk, by e-mail or any other means, it is possible that the file gets scrambled for one reason or another. The aim of chapter 2.2 was not only to explain such problems, but also to offer possible solutions to them.

In chapter 2.3 I briefly introduced different options for storing the information and taking backup copies. What is the best way for storing a corpus depends on the nature of the corpus itself: small corpora and large corpora have different needs, as well as corpora that are changed often and those that stay the same after they have been completed. The chapter explained the pros and cons of several different types of storage options.

The last chapter, 2.4, dealt with the acquisition of suitable software. There are many ways to get corpus software: a program can be specifically made for the project, or a ready-made program may be bought. With these, too, there are pros and cons depending on the needs and the resources of the project.

A general tendency in this part of the present paper has been to offer suggestions and general advice rather than specific guidelines. There are many reasons for this, and they have already been mentioned several times within the preceding chapters. On one hand, the development of computers is too fast to make any lasting decisions about what is the best option; on the other hand, the characteristics of the specific corpus and the needs of the researchers also dictate what kind of computing solutions are needed. Hopefully, however, the principles presented here will be of use to corpus compilers.

Part Three of the present paper will move on to yet another large topic that needs to be discussed in case of computer corpora: annotation. Part Three will explain what annotation is, and introduce several different types of annotation. In the beginning, it also considers the matter of getting the texts into a computerized form, as well as general annotation practices and standards.

3 ANNOTATION

So far, the present paper has not discussed to a great extent the contents of the corpus files. It has been noted that they are usually in a plain text format, or maybe in a format that has been specifically designed for the corpus. The files contain, of course, the texts themselves, which can then be browsed by suitable software.

In some cases, the plain texts are enough; however, in other cases, more information is needed. In addition to the original texts, a corpus may also include other types of information. Whatever contents the additional information actually has, it is usually inserted to the corpus in the form of **corpus annotation**.

3.1 Introduction to annotation

Many corpora nowadays are what may be called **annotated corpora**. This means that the corpus consists of not only the corpus text itself: instead, additional markup, or **annotation**, has been added to it.

Annotation can serve many different purposes. Thus, there are many different kinds of annotation that may be added to a corpus. This part of the present paper deals with the kind of annotations that are mostly seen in monolingual corpora. However, many (or all) of these annotations could also be applied to bilingual corpora, at least to each of the languages separately.

The term **corpus annotation** can be used to refer to two slightly different things. First of all, it can mean the process of adding markup to a corpus; secondly, it can mean the resulting tags in the corpus (Leech 1997a:2). In the present paper, "corpus annotation" is used in both senses, and the context, hopefully, tells the reader which one is referred to.

Adding annotation to a corpus has several purposes. Annotation brings interpretative, linguistic information to the corpus, and thus makes features of the text, such as word classes, explicit (Leech 1997a:2, McEnery & Wilson 1996:24, Hockey 1998:107). It can also be used to identify areas of the text (for example, one could search from newspaper headlines only) and for finding out where the retrieved words came from. For example, if a corpus is divided into

several subcategories (eg. newspaper articles, novels, etc.), annotation can help to determine how many instances of the search word were found in each category (Hockey 1998:107-108).

In general, annotation helps to retrieve and analyse information from the corpus and makes searches faster and easier (McEnery & Wilson 1996:24). As all corpora, annotated corpora, too, challenges our intuitions about language. Syntactically annotated corpora, for instance, may reveal that very common grammatical structures in real texts are not often presented by grammar textbooks, or that textbooks tend to concentrate on a few structures that form only a small portion of the whole range of possible constructions (Sampson 1991:183). Annotation is also very important in natural language processing, for example for developing machine translation, intelligent data-retrieval systems and voice-driven typewriters (Sampson 1991:182-183).

Annotation can include both textual and extra-textual features. In other words, in addition to codes that apply to separate words or sentences, there may be markup that applies to the whole text. These would include notes about the author of the text, title, variety of language, broad subject domain, and are likely to be included in a separate header for the text (McEnery & Wilson 1996:30).

Of course, one can also question the need for annotation. For example Sinclair (1991:21-22) has criticized corpus annotation for forcing a certain view of grammar on corpora. Especially, a point to consider is that the models of analysis that are applied to corpora are often based on more or less intuitive models of grammar that the corpus could prove to be wrong; therefore, adding such annotation to a corpus is a bit of a contradiction in terms, since it may deprive the corpus of the advantages that it inherently has. Indeed, a plain text corpus has the advantage that it includes no-one else's opinions about how the text should be seen: researchers can view it as they will.

However, for solving many research problems annotation is more or less essential. The uses for unannotated corpora are in many ways limited (Hockey 1998:108). For example, if you want to look at the auxiliary verb *will*, a simple search (with search word *will*) from a plain text corpus will certainly yield many auxiliary verbs. In addition, however, you will probably get many

instances of *Will* as a proper noun (short for *William*), and also as a common noun ("a testament"). In a similar way, if you want to, say, examine the way clothing is referred to, you might search for *trousers* – but at the same time, you might get overwhelmed by trying to think of all the possible synonyms that might be used instead (eg. *pants, slacks, shorts, leggings*) (cf. Wilson & Thomas 1997:53). Barnbrook (1996:82-83) also mentions the same problem: concordancers tend to include either too little or too much in the search results. For these kinds of problems, corpus annotation can prove to be the answer.

It should be remembered that the presence of annotation should not be seen as hindering one's own opinions and interpretations of the text. It is often possible to search an annotated corpus like a plain text corpus, if needed. One does not have to use the annotations. For example, in a grammatically tagged corpus one can specify a search for the instances of *book* as a singular noun, but there is no reason why one could not retrieve all the other forms of *book* as well, just as from an annotation-free corpus. The possibilities for different types of searches are limited by the format of the corpus, though. For example, if the corpus is not stored in plain text but in a relational database, the searching process is somewhat different.

The most common corpus annotation is the addition of **part-of-speech** tags. This means that each word gets a tag that indicates which word class it belongs to. The second most common is syntactic annotation, which is often referred to as **parsing**. Other types of annotation include, for example, **semantic**, **anaphoric**, **stylistic** and **prosodic** annotation. Each of these will be considered below. It may be noted already at this point, however, that annotations are to a great extent cumulative: the most common annotations, such as part-of-speech-tagging and syntactic analysis, are often the basis for other annotations (Leech, McEnery & Wynne 1997:100-101). It may also be questioned whether the application of these different levels of annotation should be separate processes at all, since some of them could be combined (e.g. Fligelstone, Pacey & Rayson 1997, Leech 1997b:19). These questions will be addressed in more detail in the following chapters.

3.1.1 How to add annotation

The process of annotation may be either automated or manual. In automatic annotation, a computer program is used to insert most or all of the tags. Purely manual annotation means that all tags are inserted by human annotators, in a word processor or other editor, or in a program that is made especially for inserting tags to a corpus. Often the full process of annotation is a combination of the two: a program first inserts tags, and then human annotators go through the tags and correct possible errors. It is also possible to use a program that helps the human annotator to insert the tags. For example, the program suggests what tags are possible, and the annotator selects the correct one.

To what extent annotation is automated or manual depends to a large extent on the kind of annotation that is being added. For some annotations there are fairly well-tested software that make very few errors (e.g. part-of-speech annotation), whereas some other types of annotation are still quite new and without established practices, and therefore have to be done fully manually (e.g. stylistic tagging). The issue of manual/automated tagging will be considered in more detail in connection of each type of annotation.

3.1.2 What annotation looks like

Over the years, there have been many different kinds of annotation schemes, and as Hockey (1998:108) notes, most of them are not compatible with each other. It seems that for a long time, there were no standards in corpus annotation, and each research group developed annotations that worked for their corpus and their research purposes only. This section will shortly look at the kinds of tags that one is likely to encounter in annotated corpora.

Note that nowadays most new corpus projects go for TEI tags in the first place (see chapter 3.4 below). The reason why other schemes are introduced here is that they have been much used in older corpora, and therefore one may encounter them quite frequently.

A typical way to add annotation is to attach tags to the end of the word that they describe. Thus, for example, a word like *university* could be grammatically tagged in the following way:

Example 7: Tagged word

university_NN1

"NN1" indicates that the word is a singular common noun (McEnery & Wilson 1996:36-37). This type of annotation has been used in many corpora. For example, it was used for a long time at the University of Lancaster, in their popular and highly successful CLAWS part-of-speech tagging system, a computer program which automatically assigns part-of-speech tags to text (Garside & Smith 1997:102, McEnery & Wilson 1996:36-42). The following is an example of text tagged by CLAWS:

Example 8: CLAWS tags

hospitality_NN is_BEZ an_AT excellent_JJ virtue_NN ,_, but_CC not_XNOT when_WRB the_AT1 guests_NNS have_HV to_TO sleep_VB in_IN rows_NNS in_IN the_ATI cellar_NN !_!

(http://www.comp.lancs.ac.uk/ucrel/annotation.html)

In this example, it can be seen that each word has been given the relevant tag: for example, *hospitality* is tagged "NN" meaning that it is a noun, *sleep* is tagged "VB" (verb) and *excellent* is tagged "JJ" (adjective). It may be noted that the punctuation marks, too, have been tagged (comma and exclamation mark).

One of the first corpora to use such tags was the Brown Corpus in the early 1970s, tagged by an automatic tagger called TAGGIT. Later on such tags have been used, for example, in the LOB (Lancaster-Oslo/Bergen) Corpus, which was tagged by CLAWS during the 1980s (Garside & Smith 1997:103,108).

Another type of older tags that has been widely used is **COCOA references** (McEnery & Wilson 1996:26-27, Hockey 1998:108-111). A COCOA entry consists of two angle brackets which contain a code for a variable and its value. For example,

Example 9: COCOA reference

<A Charles Dickens>

means that the author of the text is Charles Dickens, the capital *A* in the entry standing for *author* (McEnery & Wilson 1996:26-27). COCOA tags can be, however, only used for coding certain types of information, such as authors and titles (McEnery & Wilson 1996:27). Another problem is that COCOA annotation tends to be somewhat inconsistent. For example, many kinds of markup may be used for the same things, while documentation explaining the logic behind their use may not be available (Hockey 1998:110-111).

The need to be able to tag more than some predefined features of a text and the demand for consistency may be helped by the emergence of the TEI guidelines (McEnery & Wilson 1996:27). Many older corpus tagging schemes (such as CLAWS) have been updated to conform to TEI and SGML (Garside & Smith 1997:109), and new corpus projects definitely should consider going for it in the first place. Although TEI is still under development, it is based on an existing standard and has several advantages over earlier annotation schemes (Leech 1997b:30-31). SGML and TEI will be further discussed in chapter 3.4.

3.1.3 General guidelines for annotation

Although the annotation of different levels of text may be somewhat different processes, as will be seen in the following chapters, there are general guidelines that can be extended to any kind of annotation. Geoffrey Leech (1993:275) (also quoted in McEnery & Wilson 1996:25-26, and available at http://www.ling.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2maxims.htm) has defined **seven maxims of annotation** that any corpus compiler should keep in mind. These are quite general suggestions that can be applied to the compilation of any type of corpora.

1. **It should be possible to remove the annotation from an annotated corpus and revert to the raw corpus.**

This maxim has to do with recoverability: one should be able to revert the annotated corpus back to a plain text corpus, that is, remove all annotations. For a corpus that has simple TEI annotation this is not particularly hard to do. Even word processors nowadays are able to remove all tags that start with "<"

and end with ">", as TEI tags usually do. Also entity references such as *ä* are easy to convert into the characters they denote – in this case, *ä*. (cf. McEnery & Wilson 1996:25.)

2. **It should be possible to extract the annotations by themselves from the text.**

This is the "flip side" of maxim number one. One might want to, for example, store the annotations in a separate file for other use, or present them in an interlinear format so that the tags are on a separate line below running text (Leech 1993:275, McEnery & Wilson 1996:25).

3. **The annotation scheme should be based on guidelines that are available to the end user.**

There should be a manual for the users of the corpus which explains what each of the tags means and why certain tagging decisions have been made. It is a great help to the users when they do not have to try to guess what each of the tags is for, or why a particular tag has been chosen in controversial cases (McEnery & Wilson 1996:25).

4. **It should be made clear how and by whom the annotation was carried out.**

Annotation can be either automatic or manual, and there may be many people and programs involved in it. The documentation to the users should also explain this side of the tagging process (McEnery & Wilson 1996:25). It may, for example, help users to understand why some kinds of errors exist in the corpus, since human annotators and computers tend to make rather different types of mistakes.

5. **The end user should be made aware that the corpus annotation is not infallible, but simply a potentially useful tool.**

In Leech's own words, "There can be no claim that the annotation scheme represents 'God's truth'". The annotation is offered only because it is expected that many users will want to use a previously annotated corpus rather than

devise their own annotation scheme (Leech 1993:275). Corpus annotation always imposes an interpretation on the corpus: different interpretations might be just as well possible. For example, in syntactic annotation, there may be several different ways to draw a phrase structure tree for an ambiguous sentence. The corpus might represent only one of them, but this does not mean that the others are wrong.

6. **Annotation schemes should be based as far as possible on widely agreed and theory-neutral principles.**

One should not choose a little-known, uncommon linguistic theory as the basis of the annotation, unless that is relevant to the research problem. A corpus will have much more uses if it is based on some generally agreed principles. For example, for syntactic analysis, it is useful to try to divide sentences into widely-agreed categories (such as noun phrase, verb phrase), rather than try to use a rare and relatively unknown model (cf. McEnery & Wilson 1996:25-26).

7. **No annotation scheme has the a priori right to be considered as a standard.**

Only experience and general agreement among corpus researchers will yield anything that could be considered a standard. No annotation scheme is automatically better than all others, unless it has proved to be applicable to different kinds of corpora and research. (See also McEnery & Wilson 1996:26, Leech 1993:275.)

The above maxims are particularly useful because they can, in fact, be applied to a corpus project of any size. In a huge project involving maybe dozens of compilers and users, it is clear that for instance manuals and other documentation are necessary. It may not be so clear at first glance why such is needed if someone is compiling a small corpus for their own use. However, even if a corpus is in the beginning only a personal project, with time other researchers may like to use it, too. Even when there never will be other users, documenting the procedures of the compilation may prove useful for others who later want to make their own small corpus. Of course, it may prove

important to the original compiler, too, if there is any need to expand the corpus afterwards. The size of documentation for such a corpus does not need to be huge; jotting down some main principles, experiences and problems is always worth it.

Using an annotation scheme that conforms to the previous maxims also has the advantage that later on it may be easier to make the texts part of a larger corpus, if desired. It also enhances interchangability of resources, such as corpus programs and the corpora themselves.

Before the present paper moves on to introduce the actual annotations, there are three chapters that deal with more general matters. Chapter 3.2 tells about how to get the texts in a computerized form in the first place, for example by scanning or keying. Chapters 3.3 and 3.4 both deal with the matter of standardization: they talk about standards in general and about an already existing text encoding standard, respectively.

In chapters 3.5 - 3.10, I will introduce several types of annotation, starting from the most common ones, and proceeding to the rarer types. The first one to be dealt with is **part-of-speech annotation**, which is also the type of annotation that is most often encountered. The second to be discussed is **syntactic annotation**, also known as **parsing**, which is also quite popular. Yet another types are **semantic** and **anaphoric annotation**, both of which will also be introduced. After that, there will be a brief explanation of **prosodic** and other **spoken language annotations**, and some of the rarer types of annotation will be mentioned. In the end, chapter 3.11 will introduce the concept of **alignment**, which concerns bilingual and multilingual corpora.

The following chapter, however, considers a question that is rather important: how to get texts and how to convert them to a computerized form. The following chapter will tell about scanning, keying, text archives, CD-Rom collections and proofreading.

3.2 How to get texts

This chapter describes the problem of acquiring texts and getting them in the computer. In many cases, the material for the corpus is taken from books, newspapers or other printed publications. In that case, the texts usually need to be either scanned or keyed to get them into computerized form. Sections 3.2.1 and 3.2.2 tell about scanning and keying. On the other hand, there are also text archives in the Internet and commercial CD-Roms where one can find plenty of material for corpus research. Sections 3.2.3 and 3.2.4 tell about these. The last section of this chapter (3.2.5) is about proofreading, which in many cases needs to be done before the computerized texts can be processed any further.

3.2.1 Scanning

Nowadays, scanning is usually the fastest and easiest way to get a text into the computer. Scanning requires special hardware and software: an optical scanner and character recognition software (Burnard 1992:3, Barnbrook 1996:30). The scanner is the machine that is connected to the computer and reads the pages, and the character recognition software is the program that makes sense out of what the scanner reads.

There are different kinds of scanners, everything from small hand-held devices and desktop scanners to large free-standing units. The smallest ones are fairly cheap, but very slow to use and fairly inaccurate (Barnbrook 1996:30). The largest ones, on the other hand, can produce very good results.

Historically, one of the most popular scanning systems was the KDEM (Kurzweil Data Entry Machine), which was introduced in 1978 (Kurzweil 1998). The KDEM was, indeed, one of the first applicable scanning systems and it has been used successfully in many corpus projects. For example Barnbrook (1996:30) mentions that the KDEM is both fast and accurate. References to the KDEM can still be found in many articles that concern corpora. One of its advantages was that it could be trained to read particular typefaces, so that it was not limited to reading only certain kinds of fonts, like

most of the machines at the time (Kurzweil 1998, CETH newsletter 1995[1]). Burnard (1992:3), however, notes that nowadays at least the first models of KDEM seem slow and expensive. Far less expensive systems are available now that produce much better results.

A4-sized desktop scanners are both fast and accurate enough for the compilation of a small corpus. As the CETH newsletter (1995) mentions, although desktop scanners are much newer, the character recognition process that they use is usually not better that that of the KDEM. Many of them cannot be trained to read specific typefaces, either. Desktop skanners are, however, much cheaper than larger free-standing machines (CETH newsletter 1995).

Character recognition software is needed for the computer to make any sense out of the image it gets from the scanner. When scanning, the computer first inputs the text as a visual image. Then it tries to recognize the individual characters using pattern-recognition algorithms, thus converting the picture into text. The recognition of the characters is often referred to as **OCR**, optical character recognition. (Barnbrook 1996:30, Burnard 1992:3)

Johansson et al. (1996) claim that a scanner is able to distinguish highlighted text, such and bold and italic typeface, and therefore the possible tagging of such features could be done more or less automatically at this stage already. However, although it is true that modern scanning software is able to recognize highlighted text, the accuracy may not be very good. Sometimes programs fail to recognize some highlighted text, and curiously tend to see italics and bold text where there are none.

All scanners tend to make errors (Barnbrook 1996:31-32). The errors often have to do with characters or character combinations that closely resemble each other. For example, *rn* may get mixed with *m*, or *cl* with *d*. Depending on the typeface, the errors may be either occasional or fairly consistent. Hockey (1998:106) notes that sometimes there is a pattern to the errors, so that many of them can be corrected quite easily. For example, the find and replace function of word processors can be used, or suitable macros.

---

[1] The full reference for this is http://www.ceth.rutgers.edu/Ceth/newsletter/news31/ocr.html. The CETH homepage can be found at http://www.ceth.rutgers.edu. CETH is short for Center for Electronic Texts in the Humanities.

Burnard (1992:4) also notes that texts that have plenty of different typefaces in them tend to produce more errors than one with consistent typeface. Hockey (1998:106) agrees with this, and gives the example of dictionaries. One dictionary entry usually has many different font faces and thus makes it hard for the scanner to read. Burnard (1992:4) also notes that an OCR system only recognizes what is on the page; it does not know what different parts of a page actually are, and so it cannot tell running headings on the top of the page, or footnotes, apart from the rest of the text. Modern OCR software, however, offers the user the choice of the area which will be recognized; although the scanner scans the whole page, the user can select which parts will be recognized, in which order, and which can be ignored. The CETH newsletter (1995), however, notes that with longer texts this can be both "time-consuming and monotonous". They also note that OCR systems often save the text in a common word processing format, so that when opened in a word processor, the text may be reformatted to a certain line length automatically. This can be a problem especially in case of poems and other material, where correct line breaks should be preserved.

It is worth noting that the American and/or British versions of scanner software may not be able to recognize characters outside the English alphabet. For example, the program may not know the characters *ä* and *ö*, but recognizes them as *a* and *o*, respectively. When acquiring software for scanning text, one should always make sure that the program can deal with the characters of the language that is going to be scanned. Hockey (1998:106) notes that many OCR systems include a dictionary, so that the scanner can check whether some word exists or not. However, this does not help with foreign languages, and may not work quite properly anyways.

It is also worth noting that nowadays the best desktop scanners can scan images with huge resolutions. That is, they can reproduce very detailed images of the scanned page or picture. Therefore, if a good scanner is used, the errors in the scanned text are not likely to be caused by the hardware. The problem may be in the scanning software; it simply does not recognize some characters properly, maybe due to an unusual typeface in the text. However, it is also likely that errors are caused by the poor quality of the page that is

scanned. The scanner cannot tell apart text from ink plots or other smudges on the page, and tries to recognize them all as characters. Library copies of older books may also have underlinings, notes and doodles by previous readers, which invariably result in poorly scanned text. Hockey (1998:106) and CETH newsletter (1995) note that in old printed books, the baseline may be uneven. This means that the characters in one line are not quite on the same level, as some may be a bit higher or lower on the page than others. Such texts are not usually suitable for scanning. Hockey (1998:106) also notes that in the case of newpapers, poor paper quality and uneven inking make them hard to scan.

Scanning software should offer the possibility to adjust the brightness and contrast of the scanned picture, and such adjustments may indeed enhance the quality of the text the program produces. However, this does not always help. As unusually soft paper may be hard for the scanner to read, and it may be a good idea to take photocopies of the page first and try to scan it then. This is particularly true with older books, which are printed on yellowish soft paper. The CETH newsletter (1995) points out that inferior quality of the paper causes the ink to "bleed", so that the characters are not as sharp as on smooth paper. A high-contrast photocopy that has black text on bright white paper may be much easier for the scanner to read. In the case of very thick or otherwise big books it also may be easier to copy it first and then scan from the copies. Of course, one must bear in mind that a copy is always a copy; in most cases, the original page is the one that scans better.

Hockey (1998:106) also notes that scanning is rarely as successful as is expected. In fact, as she points out, "an advertised accuracy rate of 99.9 per cent means approximately one character error every ten lines". In fact, many large companies that need to get texts computerized do not use scanners at all, but have the data keyboarded professionally (Hockey 1998:106-107, CETH newsletter 1995). Keying in the data tends to produce more accurate results than scanning. Keying will be discussed further in the following section.

3.2.2 Keying

Sometimes, it is not possible to scan the needed material. Such cases include, for example, hand-written texts, unusual or very old typefaces, bad condition of

the original printed text, and obviously, spoken language (Burnard 1992:4, Barnbrook 1996:32). In such cases, the material needs to be typed into the computer by hand.

Keying takes much more time than scanning the material. However, if the person doing the typing is well skilled with the keyboard, the result has far fewer errors in it than a scanned text would (Barnbrook 1996:32). Burnard (1992:4) also notes that in some cases, such as spoken language material, some kind of transcription is probably needed anyway, whether there are plans to analyze it by the computer or not. If the transcription is something that needs to be done no matter what it is used for, it may just as well be done by computers in such a way that the material can be used in a corpus, too.

Barnbrook (1996:32) also mentions that there are companies, commercial data entry firms, that can do the typing. Using them, however, can prove to be very expensive. To achieve a high accuracy, they may use a method where the whole data is typed in twice; the two versions are then checked against each other and all differences are corrected for the final version. This procedure increases the cost of the service. Burnard (1992:4), however, says that when the whole process of compiling a corpus is considered, the cost of keying is not much compared to some other stages. Editorial costs, such as proofreading and checking the markup may prove far more expensive.

Of course, both Barnbrook and Burnard are talking about a situation where the compilers of the corpus have quite good resources, such as money, at their disposal. Researchers making small corpora for their own use may end up doing the keying themselves. It is always worth it to try to scan a text if it looks at all like something that the computer could read. The researcher will have to make the decision between keying and scanning, depending on the quality of the result. Sometimes re-typing the whole text is much faster than correcting all the errors that the scanner makes.

As to spoken language, in the future it may be possible to directly feed spoken language into the computer, so that the computer would automatically turn it into text. However, nowadays, speech recognition systems are not anywhere near good enough to do it properly (Barnbrook 1996:33). The

computer has many problems with spoken input, such as identifying the linguistic units in speech, and ambiguity of sound units. For example, it is very hard for the computer to tell apart different words in an utterance, and some utterances, such as "the old display" and "the oldest play" may indeed sound the same (Barnbrook 1996:145-146).

Keying or scanning a number of texts yourself is not, however, the only way to get usable corpus material. Hockey (1998:101) notes that most existing electronic texts are in the hands of research institutes or individual researchers, and many of such texts are neither available nor suitable for other use. Nevertheless, there are many other sources for electronic texts: plenty of them are available through the Internet, or sold on CD-Roms.

3.2.3 Text archives

Text archives, collections of texts, offer lots of material in electronic form. Many text archives are accessible through the Internet so that the material can be directly downloaded from the archive to any computer. Barnbrook (1996:26) mentions that the number of such archives has increased greatly in the recent years, and increases all the time. It should be easy to find such archives through Internet search engines, for example using "text archive" or "electronic texts" as a search word. Burnard (1992:5) also mentions that the European Commission have taken an interest toward corpora, and has founded a survey of texts and other sources that are available in Europe.

The material in text archives is usually free to use. Sometimes there may be some limitations mentioned, but those often have to do with commercial uses (Barnbrook 1996:26). All text archives should include a notice which explains the terms of use of the material and possible restrictions. Before downloading any material, one should look for such a notice and read it through to be sure that the texts can be used for the desired purpose.

Sometimes, the texts cannot be downloaded, but they may be accessed directly for research purposes (Barnbrook 1996:27). Such access may be achieved through the Internet, or sometimes by telnet connection. There may be a limited access for occasional users, e.g. the number of searches is limited, or the program prints only a limited number of the resulting sentences. To get

full access, one must register as a user, and be prepared to pay something for it. Examples of such services on the Internet include Collins Cobuild's Cobuild*Direct* (http://titania.cobuild.collins.co.uk) and the W3 Corpora site (http://clwww.essex.ac.uk/w3c/) (http addresses were valid September 1998).

Even for registered users, there are likely to be limitations as to what kind of research can be done, because usually only the software at the host site can be used (Barnbrook 1996:27). If one wishes to examine the material with some other program, it is not likely to be possible.

Although text archives may offer an overwhelming amount of texts, their suitability for a particular research question may not be very good. Barnbrook (1996:26) notes that if a corpus is compiled from text archives only, it will be seriously limited by the range of texts in the archives. He mentions that the archives mostly include literary texts from particular historical periods, and a quick look at the Internet seems to confirm this. This probably has to do with copyright issues; since the copyrights of older texts have expired, and texts such as folklore and religious writings do not usually have copyright holders at all, anyone can distribute them.

It is worth noticing that most book publishers nowadays have their published material in electronic form, too. Since they use word processing and desk-top publishing tools, a computer-readable version exists. One may be able to get them, but it is possible that the publisher charges for them and places restrictions on their use (Barnbrook 1996:29). Barnbrook also notes that many academic institutions have academic writings in electronic form. For example, at the University of Jyväskylä, there are plans to make all new theses available through the Internet (http://docuweb.jyu.fi). Such academic writings should be freely available.

3.2.4 CD-Roms

In addition to texts that can be downloaded through computer networks, plenty of texts are available on CD-Roms, too. There are many text collections, texts from the archives, and newspapers and magazines that can be bought on CD-Rom (Barnbrook 1996:28, Burnard 1992:5). Publishers are becoming more aware that there is a demand for electronic texts, and among others, the

newspapers Guardian and Independent have published their material on CD-Roms (Burnard 1992:5). Many large corpus projects, too, distribute their corpora on CDs (Burnard 1992:5), sometimes the full corpus, sometimes parts of it (Barnbrook 1996:28). One of the distributors is ICAME (http://nora.hd.uib.no/icame.html), whose CD-Roms include many well-known corpora, such as the Brown, LOB and London-Lund Corpora. As the capacity of ordinary CD-Roms nowadays is around 650 MB, one disk can include huge amounts of text (Barnbrook 1996:28). By buying even one of such collections, one may get more or less exhaustive quantity of material for corpus-based research.

The material on a CD-Rom, however, may be stored in a format that cannot be used by any other program but the one that comes on the disk itself. As Hockey (1998:104) notes, most electronic texts were originally created for specific research purposes, so that it is hard to use them for anything else as such. Even if it is possible to easily convert the files to another format, the licencing agreement, i.e. the terms of use of the disk, may prohibit it (Barnbrook 1996:28-29). Before buying any CD-Rom collections, one should find out how the data is actually stored on the disk; can it be used in other programs as such, should it be converted first to another file format, can it be converted by the tools that are available, and is it permitted to do any of that.

For example, if the disk is designed to be used through an Internet browser (for instance a collection of newspaper articles that is not actually meant for linguistic research) it will be in ASCII format that is HTML tagged. There are, however, plenty of tools to remove the HTML tagging – indeed, a word processor such as Word 97 has it as a build-in feature. Therefore, if the licensing agreement allows it, it will be fairly easy, although time-consuming, to convert the material back to plain text. In other cases, however, it may be much harder. For example, if the data has been processed into a special kind of database, it is practically impossible for a new user to convert it back to ASCII.

If texts are acquired from the Internet or from CD-Rom, there should be no need to proofread them anymore, unless the removal of undesired tags is considered proofreading. However, with keying and scanning, proofreading is the logical next step.

3.2.5 Proofreading

After the text has been computerized, it should be proofread. Both keying and scanning are likely to leave the text with errors in it. To get the best possible results, the text should be proofread manually, in other words, by human annotators (Barnbrook 1996:38). It is a fairly time-consuming task and even after that, the text is not necessarily error-free. It is quite easy for the person doing the proofreading to become blind to the errors, that is, not notice them after staring at the text for a while. Just as the scanner software confuses characters that look like each other, a human reader confuses characters and words that resemble each other. As Barnbrook (1996:38) says, "its [=proofreading] usefulness will be entirely dependent on the skill and reliability of the proof-reader". The person doing the proofreading needs to be proficient in the language that is being proofread, and able to do the job so that the time spent on it and the accuracy of the resulting text are acceptable.

One way to make proofreading easier and more accurate is to change the font to `courier`, or rather `courier new`. They are both very clear and the characters stand well apart from each other. The fonts that are used normally, such as Times New Roman or Paladino, tend to draw the characters fairly close to each other. Compare, for example, *ri* and *n* with Courier New `ri` and *n*, which are far easier to tell apart, even more so on the screen than when printed.

Nowadays, all major word processors include a spell checker. One should definitely take advantage of it. The computer is much better than human beings at spotting scanning errors and mistypings; *rcturn* and *return* may look the same to a human proofreader, but the computer can definitely tell them apart. What the computer cannot do, however, is distinguish words that are mispelled, but still look like correct words. Barnbrook (1996:37) gives an example of the word *tool*. If it is spelled as *tolo*, the computer will notice it, but if it is spelled *toll*, the computer will not take note of it because it still looks like a correct English word.

The computer's proofreading facility works on basis of a word list, and considers all words outside the word list as possible errors. Therefore, an automatic spell checker usually gives quite a number of false alarms. For

example, many proper names will be seen as errors, as will be all unusual variation in spelling. Spell checkers also tend to use, in case of the English language, only one variation of English at a time (e.g. American English or British English). Such spell checkers report as errors all the deviations from their own version of how a word should be spelled (e.g. *theatre – theater*). Many spell checkers are, however, able to learn new words. It means that words can be added to the word list, or the users can define word lists of their own.

It is usually not necessary to try to weed out every possible error in the texts. As Barnbrook (1996:37) says, one should determine an acceptable level of error before deciding how carefully the text needs to be proofread. Depending on the kind of research that the data is for, certain amount of errors may be acceptable. In other words, a few errors in a huge corpus are not going to distort the research results significantly.

This chapter has examined the matter of getting material in computerized form. In many cases, the material has to be scanned into the computer; in some cases, scanning may be impossible, so that the texts have to be keyed in. There are also text archives and CD-Roms available that have plenty of texts in them. The advantage with them is that they are in computerized form already, and they do not have to be proofread anymore. However, the material in such collections may not be suitable to specific types of research. The end of this chapter, then, gave some advice about proofreading.

After the texts have been fed into the computer one way or another, and proofread, they may be ready to be used in a corpus. However, there are many cases when plain texts are not enough. In might be useful to have some kind of annotation in them that eases the corpus searches one way or the other.

The rest of this part of the present paper deals with the matter of annotation. Before going on to introduce specific types of annotation, however, the following two chapters examine annotation practices on a more general level. In order to make the interchange of material and programs possible, there has to be some standards for annotation. The following chapter is about the need for standards in general, and chapter 3.4 introduces the basic principles of

SGML and TEI. SGML (Standard Generalized Markup Language) is used in the creation of structured documents; the Text Encoding Initiative (TEI), on the other hand, has developed guidelines for corpus annotation that in the long run might become a standard in corpus work.

3.3 Standards

Before going on to describe the various kinds of annotations that can be applied to corpora, it is necessary to have a look at the question of standardization. In the past, many corpus projects have each used their own forms of annotation, suitable for their own purposes. However, during the last decade or so there has been movement towards common guidelines and standards for annotation.

There are several reasons why annotation practices should be standardized at least to some extent. One of the most important of them is re-usability (Leech 1997a:7): the same corpus can be used again by different researchers, in different programs, and may be combined with other corpora. When the annotation used is the same, the material in different corpora can be interchanged and used for many different purposes. As Kahrel et al. (1997:232) note, corpora have been created in several different countries, both in the same language and several different languages. If these corpora conform to some common guidelines, it is easier to compare the results and continue with the work. If work in different languages conforms to the same standards, it is possible to combine them into a multilingual corpus.

Annotation standards also make it easier to start new projects. When a well-documented annotation practice already exists, new corpus projects do not have to start from the beginning; instead, they can use what already exists and what they may already be familiar with from other projects (Kahrel et al. 1997:232, Leech 1997a:7). They may also take advantage of the experiences of those who have used the same annotation scheme before, and are able to avoid the mistakes that may have been made earlier.

Standards also enable researchers to use the same tools for different corpora (Kahrel et al. 1997:232). This includes both annotation tools and corpus browsers and concordancers. As McEnery & Wilson (1996:174-175) note, the recent interest in SGML-tagging (which will be explained in more detail in the following chapter) has already put pressures on software developers to provide suitable programs for the many users who would like to have them.

Introducing standards creates some problems, however. As Leech (1997a:6-8) explains, different corpora and different projects may need rather different type of annotation. For example, some researchers need a small corpus with very specific annotation that is useful for their specific research, whereas others want to have a large corpus with only some general-purpose annotation in it. Kahrel et al. (1997:234) note that there are both corpora that serve a very theoretical purpose, such as linguistic research, and those that are meant for very practical uses, such as natural language processing (e.g. machine translation). In addition, despite the fact that most annotated corpora so far have been in English, there is more and more interest towards the annotation of other languages, some of them maybe very different from English. As Leech (1997a:8) puts it, any standard that is put forward should avoid "[imposing] a straightjacket of uniformity" and leave enough room for flexibility.

Already existing corpora also form a problem. There may be corpora of millions of words that have been tagged according to their own specific annotation scheme, and it would require much extra work to fit them into some complete new type of annotation scheme. Kahrel et al. (1997:233) suggest that any new standard should also take into account older corpora, and be flexible enough so that the older corpora can be made to fit into it without too much effort.

The guidelines would also have to be such that they have been already tested and accepted, and proven useful to researchers. It is not possible to put forward a number of rules and expect that researchers will start to use them just like that. No annotation scheme is automatically better than another one; many have good qualities, and only practical experience will bring out the best. (cf. Leech 1997a:6-7.)

There are, in fact, two levels of standardization that can be considered. One of them is standard encoding of corpora and annotations; the second is standard annotation of corpora (Kahrel qt al. 1997:231). The difference between these two is that the first one refers to the actual codes and markup that are used, whereas the second refers to what type of features should be tagged.

The first of these has been addressed by the Text Encoding Initiative (TEI). TEI suggests SGML-based tags that can be used in corpus annotation. SGML and TEI will be considered in the following chapter. The second of these, however, has been addressed by the EAGLES initiative and will be discussed below.

EAGLES (Expert Advisory Group for Language Engineering Standards) is a European effort to give some common guidelines for text encoding (Kahrel et al. 1997:231, 235). EAGLES has examined existing annotation practices in Europe, and formed specifications for annotation that are to be used in future EU funded work (McEnery & Wilson 1996:29). They have suggested some rules for morphosyntactic and syntactic annotation of corpora. They explain, for example, what kinds of tags are obligatory, recommended or optional for morphosyntactic annotation: for instance, any part-of-speech tagset should have tags for nouns and verbs, that is, those tags are obligatory for any part-of-speech tagset (Kahrel et al 1997:235). They also give standards for the documentation of annotation (Kahrel et al. 1997:240-241): what kinds of things the documentation should include and to what extent and accuracy they should be reported. The EAGLES guidelines, on the other hand, provide rules to be applied to any European language, but at the same time they are flexible enough to accommodate differences between languages (McEnery & Wilson 1996:29).

The following chapter, however, concerns SGML and TEI, which provide a corpus compiler with tags that can be used in corpus annotation. The chapter also explains why such tags are technically necessary; why, for example, word-processor-made markings will not do for annotating the corpus. The chapters after that, then, will go on to explain corpus annotation further, and deal with the reasons why a researcher would want to have any annotation in a corpus in the first place. It may be noted here already that not all of the annotation systems presented later on conform to any standards. That is mainly because they are used on older corpora, or conversely, deal with new innovations for which no standard exists yet. However, any new corpus project should look for some existing guidelines, and the TEI guidelines are a good start.

3.4 SGML and TEI

Any normal text contains far more textual information than simply alphabetic characters following each other, forming words and lines on paper. There are sentence and paragraph divisions, page numbers, chapter headings, footnotes, highlighted text, and several other more or less distinct components that the text can be divided into. A chapter may be devided into sections, which further may have their subsections; a poem may be divided into stanzas and lines, and in the end, into separate words.

Most of these divisions are clear to a human reader. A person reading a text is able to tell which is the chapter heading and which is the chapter text itself, and whether a capital letter signals the start of a new sentence or a proper noun. A human reader is able to conclude that an empty line in the text is likely to mean a paragraph change, or that a new page marks a new chapter.

Some of these divisions may be clear to the computer, too, especially if the text is in a word processing format. Word processors are able to use different typeface for headers and take care of the numbering of chapters. Italics or bold face may be added for emphasis, the word processor using its own hidden coding system to keep track of them. The program can position footnotes to the bottom of the correct page and check the page numbering when new text is added. Such coding, however, is specific to the particular word processor that is used; its format can be read only by the program itself, and maybe a few other select programs. If the text is saved in ASCII format, as is often done in case of a corpus, all such special information is discarded, leaving only the plain text.

In addition, some divisions, such as sentence boundaries, are not very relevant to a word processor and it is usually not very well aware of them. A word processor is generally intelligent enough to keep an end-of-sentence period on the same line with the sentence itself, but that is about all the interest it has toward sentence divisions. The computer handles the text simply as characters following each other. Plain ASCII format is the most stupid of all formats that text can be stored into: a line break is the only division in the endless line of characters that it can recognize, and line breaks may be in weird places from a human point of view.

However, ASCII format is often the only possible choice for a corpus. Since it is (at least theoretically) system- and program-independent, it makes it possible to study the texts in several kinds of corpus browsers and move them around between different kinds of computer systems and setups.

Depending on the uses and later processing of the corpus, one may want to keep more information of the text than the plain ASCII format provides. For example, one might like to be able to tell apart normal prose text and poems which are embedded in it, or be able to search streches of text under certain kinds of chapter headings. Also, if the text is to be aligned later with its translation into another language, so that a sentence and its translation can be easily found, it makes the whole task much easier if the sentence boundaries are clearly marked.

In addition, there are many types of information that are not available in a word processing format, either. For linguistic research it might be useful to have, for example, word classes made explicit in the text. One might also want to have information about sentence structures or the meanings of the words alongside with the text itself. It would be an advantage to be able to search not only for specific words from a corpus, but for some other attributes of them, too. For example, one might want to distinguish all the instances of *bank* as "riverbank" from *bank* as "financial institution".

In addition to there being the chance to somehow mark such things, in the long run it is essential that the markings are not unique, but understandable to several programs and users, as was pointed out in the previous chapter. Although the corpus may be originally designed for a very limited use, it may come obvious after a few years that the same material could be used for some other type of research, too. Further, new useful programs may come around that might be ideal for the old corpus. If any features of the corpus need to be annotated, it is profitable in the long run to do it in a way that has more than one use to it. And, luckily, such an encoding scheme does exist, as part of the **SGML** standard.

SGML (**Standard Generalized Markup Language**, ISO standard 8879) is an international standard for representing texts in an electronic form. It is both device-independent and system-independent (Sperberg-McQueen and

Burnard 1994:13). System-independence means that a text with SGML markup is not restricted to any one operating system or hardware. Device-independence means not only that such a text can be used in any kind of computer, but also that in addition to being printed on paper or displayed on screen, it can be output to any kind of device, such as a voice synthesizer. As Sperberg-McQueen and Burnard (1994:15) point out, the basic design goal of SGML is its data independence: SGML encoded texts should be transportable from different kinds of computer environments to others without a loss of information. Note that although SGML encoded texts are sometimes referred to as being in an "SGML format", the file format of SGML documents is actually plain ASCII text. In addition to the textual data itself, an SGML document has SGML tags in it, but it is all stored in plain text format.

SGML provides a means to add special markup, or encoding, to any text. Sperberg-McQueen and Burnard (1994:13) define markup as "any means of making explicit an interpretation of text". As they point out, in a way all written texts are encoded thus; punctuation marks and capital letters tell the reader where sentences start and stop, and white space signals the boundaries of words. SGML could be described, in a way, as a set of punctuation marks and other special markings that are explicit to the computer just like a capital letter, or bold face text, is to the human eye. SGML "punctuation", however, has a lot more to offer than simple commas and periods do.

The basic idea of SGML is that any text can be broken down into smaller components, **elements**, which form a hierarchical structure (Simons 1998:17, Hockey 1998:111-112). An element is a textual unit, such as highlighted text or a paragraph, that can be distinguished from a stretch of text. An element is marked by inserting a start tag in the beginning of it and an end tag after it (Sperberg-McQueen and Burnard 1994:16). The tags always consist of arrow brackets <>, and there is a start tag and an end tag for all elements. The end tag looks basically the same as the start tag, except that it has a slash in it after the first bracket. For example, the sentence start tag is *<s>* and the sentence end tag is *</s>*. The end tag does not have any variables in it either, which the start tag may have. For example, to denote text in a foreign language, there might be a start tag *<foreign lang=fr>* which just means that

the text that follows is in French (variable *lang,* "language", is given the value *fr* for "French"). The end tag, however, is simply *</foreign>*. In many cases, the end tag can be omitted, if it is otherwise clear from the structure of the text where an element ends. For instance, paragraph end tags can often be left out, since the beginning of a new paragraph (and a new start tag) signals that the previous paragraph must have ended.

Elements may also have relationships to other types of elements. This is what is meant by the hierarchical structure of SGML. For example, a *<poem>* may include several *<stanza>* elements, and each stanza in turn may consist of several occurences of *<line>* (Sperberg-McQueen and Burnard 1994:16-17). With this kind of hierarchical tags, it is possible to define very complex document structures.

Hockey (1998:112) also points out that the purpose of SGML tags is not to tell the computer what it should do with the element; rather, the tags tell what an element *is*. In chapter 1.2.1 there was an example of a tagged text, which had tags for paragraph and sentence boundaries. The following example includes them, too:

Example 10: SGML tagged text

```
<p><s rend=italic>"...I need... I need... I need..."</s></p>
<p><s>Frowning, Marty clicked off the recorder.</s></p>
<p><s>  His train of thought had clattered down a siding
and chugged to a stop.</s> <s>He could not recall what he
had been about to say.</s></p>
<p><s>Needed what?</s></p>

                                    (Dean Koontz: Mr. Murder (FECCS))
```

The tag *<s rend=italic>* signals the beginning of an italicized sentence. *<s>* -tags surround sentences, and *<p>* -tags mark paragraph boundaries.

In addition to elements, there are **SGML entities**. Whereas tags marking elements are comments about the structure or layout of the text, entities become part of the text itself (Sperberg-McQueen and Burnard 1994:29-30). An SGML entity starts with an ampersand (&) and ends with the semicolon (;). As was mentioned in chapter 2.2, entity references can be used, for example, to replace otherwise problematic characters such as *ä* and *ö*. The entity references *ä* and *ö* can be used to replace all *ä* and *ö*

characters in a text, thus avoiding problems with different character sets. When a program that can read SGML encoded text encounters entity references, it knows how to replace them all with the correct characters on screen or paper (Sperberg-McQueen and Burnard 1994:30). Entity references can be used to encode special characters when there are not too many of them. However, if a language uses a completely different kind of script, this would not be practical anymore (cf. Hockey 1998:112). Entity references are also sometimes used to tag the characteristics of single words, as will be seen in the following chapters (e.g. grammatical tags may take the form of SGML entity references).

What needs to be pointed out here is that SGML has much more to offer than a simple way to annotate texts for linguistic research. SGML is an extremely powerful tool for controlling structured documents and making hypertext applications. The hierarchical and structured nature of SGML is usually seen as its essence, and corpus tagging is applying only a tiny part of its possibilities. SGML is, in fact, a metalanguage that can be used to define markup (Sperberg-McQueen and Burnard 1994:13): it is possible for the user to define an unlimited amount of different types of elements and entities, and their relationships to each other.

Finally, all such definitions combined form a **document type definition (DTD)** (Sperberg-McQueen and Burnard 1994:14-15, 33-35). The DTD defines what kinds of elements and entities there can be in an SGML document. Simons (1998:18), interestingly enough, points out that "a DTD is really nothing more than a context-free grammar", so that it should look familiar to linguists, despite a different way of writing the rules. However, it may not look very inviting to linguists who are not particularly interested in phrase structure rules, and indeed there is no pressing need for corpus compilers to actually understand the DTD. Many SGML guidebooks that explain DTDs are directed at people who need to construct structured documents and need to define the structure themselves, which goes far beyond the scope of what a corpus compiler needs to know about SGML. The ordinary corpus compiler usually does not have to bother with creating or modifying DTDs for their corpus: it has been done already by the **Text Encoding Initiative**, or **TEI**.

The Text Encoding Initiative was started in 1987 under sponsorship of the Association for Computers and the Humanities (ACH), the Association for Computational Linguistics (ACL) and the Association for Literary and Linguistic Computing (ALLC). The purpose of the project was to unify existing encoding practises, simplify the processing of texts by computer and make exchange of texts easier for the humanities computing community (Sperberg-McQueen and Burnard 1994:Preface). In 1994, the project published the Guidelines for Electronic Text Encoding and Interchange (Sperberg-McQueen and Burnard, 1994). The Guidelines define a DTD which is designed for the encoding of any type of text, in any language, independent of any specific use for the texts (Sperberg-McQueen and Burnard 1994:Preface). The point is to create a uniform way to represent texts electronically and make it possible to move them easily between different computers and applications.

In practice, what this means to a corpus compiler is that the TEI Guidelines define a set of tags that are especially suitable for the tagging of a corpus. As Hockey (1998:114) explains, TEI is based on the principle that all texts have some features in common. There are about 60 elements which form the common core of TEI, which include tags for titles, lists, quotations, names and dates, among other things (Hockey 1998:114). In addition, there are many other features in TEI that can be used when needed, depending on the text type and the uses of the material. As Hockey (1998:112) points out, TEI is incremental so that another person may add tags to a text later on, and anyone can decide to tag and use only those features which are important for a given research. There are also specialized base tag sets for different types of texts, such as verse, drama, dictionaries and transcriptions of speech (Hockey 1998:114). In addition, the TEI tagset may be added to or modified as needed, although it may reduce the reusability of the material.

In the TEI, each text is seen as consisting of a header and a body. The header contains information such as author, title, date and other bibliographical information; the body consists of the text itself (cf. McEnery & Wilson 1996:27, Hockey 1998:114).

It would seem advisable for new corpus projects to use TEI-conformant tagging schemes from the beginning. Since TEI is still a rather new

development, it has not been used extensively and there are not many tools that could take advantage of it yet. This should not, however, prevent people from using it. According to Hockey (1998:114-115), it is likely that more SGML/TEI-based software will soon be available for linguistic reasearch. She also points out that TEI helps with many problems that were apparent in earlier encoding schemes; therefore, its use should be encouraged. There are, in fact, already several projects that have used SGML tags, for example Healey (1997), Bailey & Curzan (1997) and Wilson & Rayson (1993). Also the well-known CLAWS tagging system at the University of Lancaster has been updated to use SGML tags (Garside & Smith 1997:109). So far, the experiences with SGML seem to be positive.

This chapter has introduced SGML, which is used to describe the hierarchical nature of structured documents. In addition, this chapter introduced TEI, which is an SGML based "grammar" that can be used to annotate texts for linguistic research and corpora. The TEI Guidelines can be used to tag any features of a text, from large-scale structures such as chapter divisions, to small-scale features, such as characteristics of individual words.

The following chapters move on to describe the actual annotations that markup is used for. Note that many of the annotations introduced do not use a markup scheme that is TEI conformant, because they were designed before TEI. Rather than explaining the actual form of the markup used, the following chapters try to explain the contents, principles and goals of different types of annotations. The actual markup, then, may take many forms, which hopefully conform to an existing or developing standard.

All the following chapters have basically the same structure. First, the relevant type of annotation is introduced and some of its uses described. After that, the practical side of annotation is considered: how to add this type of annotation. In the end, the restrictions and problems of the annotation will be discussed.

3.5 Part-of-speech annotation

This chapter is about part-of-speech annotation. First, general aspects of part-of-speech annotation are explained. After that, there is a section on how part-of-speech tagging is added to a corpus, and how automatic taggers work. In the end of the chapter, the problems of part-of-speech tagging will be considered.

Part-of-speech annotation means that each word in the text is given a code which indicates which word class it belongs to. Part-of-speech annotation can also be referred to as **grammatical tagging**, or **morphosyntactic annotation** (McEnery & Wilson 1996:36). For example, in a phrase like *The red book* the words would be tagged as article, adjective and singular common noun, respectively. In a fully grammatically tagged corpus each word would have such a tag. From here on, in the present paper, part-of-speech tagging will be usually referred to as grammatical tagging, or simply as **POS-tagging**.

The following is an example of grammatically tagged corpus text from the Spoken English Corpus at the University of Lancaster. It has been automatically tagged by CLAWS.

Example 11: Grammatically tagged corpus text

Perdita&NN1-NP0; ,&PUN; covering&VVG; the&AT0; bottom&NN1; of&PRF; the&AT0; lorries&NN2; with&PRP; straw&NN1; to&TO0; protect&VVI; the&AT0; ponies&NN2; '&POS; feet&NN2; ,&PUN; suddenly&AV0; heard&VVD-VVN; Alejandro&NN1-NP0; shouting&VVG; that&CJT; she&PNP; better&AV0; dig&VVB; out&AVP; a&AT0; pair&NN0; of&PRF; clean&AJ0; breeches&NN2; and&CJC; polish&VVB; her&DPS; boots&NN2; ,&PUN; as*CJS; she&PNP; 'd&VM0; be&VBI; playing&VVG; in&PRP; the&AT0; match&NN1; that&DT0; afternoon&NN1; .&PUN;

The codes used are:

AJ0: general adjective
AT0: article, neutral for number
AV0: general adverb
AVP: prepositional adverb
CJC: co-ordinating conjunction
CJS: subordinating conjunction
CJT: that conjunction
DPS: possessive determiner
DT0: singular determiner
NN0: common noun, neutral for number
NN1: singular common noun
NN2: plural common noun
NP0: proper noun
POS: genitive marker
PNP: pronoun
PRF: of
PRP: prepostition
PUN: punctuation
TO0: infintive to
VBI: be
VM0: modal auxiliary
VVB: base form of lexical verb
VVD: past tense form of lexical verb
VVG: -ing form of lexical verb
VVI: infinitive form of lexical verb
VVN: past participle form of lexical verb

(http://www.ling.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2posex.htm)

Note that in this example, the tags that are used are TEI tags. The example is tagged by the most advanced version of CLAWS so far. In a couple of cases, there are two tags for one word: for example the first word, *Perdita*, has tags for both singular common noun (NN1) and proper noun (NP0). This is because the computer has not been able to decide which one is the correct tag.

POS-tagging was one of the first types of annotation that was applied to corpora. Still today, it is definitely the most common type of annotation (McEnery & Wilson 1996:39). One reason for its popularity is that even when it is not used for any specific purpose as such (although most often uses for it can be found), it is more or less necessary in order to perform some of the higher-level annotations, such as parsing (Leech 1997a:5, McEnery & Wilson 1996:36). Indeed, at first, during the 1960s, POS tags were seen at least by some researchers as a "side-effect" of parsing rather than useful in their own right.

There are, however, many good reasons to use POS tags. The most important is that POS tags make searching a corpus much easier. It enables the researchers to specify in greater detail what they want to retrieve (McEnery & Wilson 1996:36). For example, the word *left* can belong to several word classes: depending on the context, it can be an adjective (*my left hand*), an adverb (*turn left*), a noun (*on your left*) or a verb (*I left early*) (Leech 1997a:4). If the corpus is grammatically tagged, the researcher can take advantage of the tags and specify which instances of *left* are to be retrieved. Such disambiguation of words can also prove useful to speech synthesis, that is, computer programs that try to speak or "read" aloud. The information of whether *lead* is a noun or a verb tells the program whether it should be pronounced /led/ or /li:d/ (Leech 1997a:4-5).

Other reasons for applying POS tags to a corpus are re-usability and multi-functionality (Leech 1997a:5-6). Once the tags have been added to the corpus, it can be used many times without the need to analyse it all over again. The corpus can also be used for more varied purposes. The addition of tags does not limit the uses of the corpus; on the contrary, it brings many more possibilities. It opens up new ways to process the corpus as such, and, as noted before, creates the basis for higher level annotations.

POS tagging can also be done very accurately automatically, that is, a computer can insert the tags correctly without human help. Even in the early seventies, the TAGGIT tagger was 77% accurate, which means that 77% of the words were assigned the correct word classes. In the early eighties CLAWS achieved over 95% accuracy and today, most taggers get around 95-98% (McEnery & Wilson 1996:39, Garside & Smith 1997:102). Having the computer do the task instead of human beings saves both time and money (McEnery & Wilson 1996:126). Thus it also makes the compilation of large corpora possible within reasonable expenses.

3.5.1 How to add part-of speech tags to a corpus

POS tags can be added either automatically or manually. As was mentioned in the previous section, automatic taggers can achieve very high accuracies and they can process long streches of text in a short time. In fact, Marcus et al. (1993:278-279) compared the speed and accuracy of humans vs. computers. The result was that manual tagging took twice as long as computerized tagging with humans only post-checking the tagged data. In manual tagging the error rate was also much higher, and there were more disagreements between annotators about the correct tags.

It may also be noted that since POS tagging has been used a lot, there are conventions for doing it, and the problems associated with it are well known. It might be said, therefore, that there is a strong argument for letting the computer do most of the tagging. The tags can be later checked by human annotators, who can correct errors and deal with problematic cases that the computer has not been able to tag.

It may be noted that computerized annotation is never 100% accurate, which is due to problems that will be addressed in section 3.5.2. Any corpus tagged by a computer always has errors in it, and will have them even after human post-editing, since human beings do not achieve perfect results, either (McEnery & Wilson 1996:126, Baker 1997:243-244). As Baker (1997:243-244) notes with regard to human post-editing, humans tend to make errors, too, and lack the consistency that a computer is bound to have. Since automatic

taggers are usually used for adding grammatical tags to a corpus, the following section describes how a tagger works.

3.5.1.1 How a tagger works

To add grammatical annotations to words, an automatic tagger goes through several stages with each word. Although different taggers undoubtedly have their own slightly different methods, the description given by McEnery & Wilson (1996:120-124) exemplifies the process in general terms, and can be thought of as a basic explanation for the workings of a tagger.

The tagger has a lexicon, which includes all the possible tags for each word. Therefore, when the tagger starts to assign a grammatical tag to a word, the first thing it does is to check whether the word can be found in the lexicon. If it is, the tagger simply assigns all the possible tags to it at this point (eg. *book* gets tags for both noun and verb).

If the word is not found in the lexicon, it goes to morphological analysis. Morphological analysis tries to find out if the word has an ending or a prefix it could recognize. For example, the word *books* could be subjected to morphological analysis because the plural form was not found in the lexicon. The analyser finds a familiar ending, *-s*, removes it, and sends the resulting word (*book*) back to be checked against the lexicon. A word can be sent around between the lexicon and morphological analysis several times; if it is still not recognized in the end, the system may simply assign all possible tags (considering the context) to it and leave the decision to a later stage.

For cases where two or more words form one syntactic unit (eg. idioms and phrasal verbs), the tagger may have a special lexicon to recognize them and tag them properly. Such word combinations tend to create problems for the tagger, though: this will be dealt with in section 3.5.2.1, in connection with other problems in POS tagging.

After all of the text has been processed through, and each word has been assigned one or more POS tags, the program starts what is called the **disambiguation** process. It means that one way or another, the computer decides which one of all the possible tags is the correct one for each word. The following section briefly explains the most common ways to do this.

3.5.1.2 Rule-based, probabilistic and hybrid taggers

There are mainly two types of taggers, **rule-based** and **probabilistic taggers**. The difference is in the way they decide which tag is the correct one. A rule-based tagger, basically, first finds out which tags are possible for a word, and then uses the context (a few words around the word that is being tagged) to decide which is the correct one. To achieve the latter, it has a set of rules which test whether the tag might be the correct one, or impossible in the context (Garside & Smith 1997:103). The rules tend to be the result of linguists' intuitive knowledge or observations of data (McEnery & Wilson 1996:123, Garside & Smith 1997:103) and may not, therefore, always correspond to reality very well.

Probabilistic taggers, on the other hand, calculate probabilities for tags. Each word, on its own, has statistical probabilities of belonging to a certain word class or number of classes; on the other hand, there is the possibility of certain word classes following each other. The tagger tries to come up with a sequence of tags that has the highest probability of occuring together: for example, it is likely that an article is followed by an adjective or noun rather than something else. (Garside & Smith 1997:102-104.)

For instance, in a phrase like *the red book*, *the* is definitely an article and *red* is very likely to be an adjective (it could be also a proper noun, as in *the Reds won the match*, but the probability for this is likely to be much lower). *Book* could be either noun or verb, but since it is preceded by a very likely adjective, which in turn is preceded by an article, the tagger can deduce that in this case, *book* must be a noun. There may be a probability that either *red* or *book* could be yet something else in some special circumstances, but the chances for that are rather low.

The probabilities are, in the first place, derived from existing POS-tagged corpora (McEnery & Wilson 1996:124). A program that is meant for the task can go through an annotated corpus and get statistical information on what the probabilities are. This information, in turn, can be used by taggers in order to tag new corpora. It may also be noted that probabilistic taggers are quite good at tagging words that their lexicon does not know: the probabilities for certain word-classes occuring together can be used to decide the word-class

of an unknown word (McEnery & Wilson 1996:124). However, as can be expected, tagging accuracy is higher for words that are in the lexicon (Smith 1997:141).

It may be noted here that many taggers are able to learn more as they are used. The statistical data becomes more accurate as more texts are processed, and new words get added to the lexicon (e.g. Greenbaum 1993:12). Thus, many taggers can be trained to perform better over time. There are several opinions, though, about whether for instance a large lexicon is only for the good (cf. Sánchez León & Nieto Serrano 1997:157). It does not necessarily increase the accuracy of the tagger. Many corpus taggers seem to produce quite good results with rather a small lexicon and few rules or limited probabilistic data.

In reality, many taggers are more or less **hybrid** taggers, which means that they combine the good qualities of both rule-based and probabilistic taggers. They are often seen as the best kind of taggers and they have achieved very high accuracies. Probabilistic taggers are, however, the most common type of taggers nowadays, even though rule-based taggers have been making something of a comeback, too. (Garside & Smith 1997:102-106.)

3.5.1.3 Tagsets

One more thing to note about inserting the tags is that there must be a **tagset** defined. It is a list of all the tags that may be used. There are huge differences between different tagging schemes in how many tags are included in the tagset, ranging somewhere between 30 and even over 400. Tagsets for English tend to include 30-200 different codes (Leech 1997b:29).

In practice, there may be a problem between what one wants to have in a tagset and what is possible within reasonable resources. Leech (1997b:24-25) gives an example of this. An "armchair linguist" might very well come up with nice-looking word categories that are linguistically useful. However, it may be that it is impossible to assign such tags to a corpus automatically. For example, you can have sentences like *come what may* (*come* as subjunctive), *come here!* (*come* as imperative) and *they come every spring* (*come* as present tense plural indicative). Such distinctions of the word *come* make sense

linguistically, but are very hard for the computer to make, since in each case, *come* looks just the same and the context does not help much. It would simply take too much time to manually check and correct all the errors that the computer would make with them.

McEnery & Leech (1996:41-42), too, talk about this same problem. They mention that there is a "conflict in annotation between retaining fine distinctions for maximal utility to the end user and removing difficult distinctions to make automatic annotation more accurate". Greenbaum (1993:12-14) also notes that in designing a tagset, both the annotators' and the researchers' needs should be taken into account. For human annotators, too, a very fine-grained tagset may prove problematic, since there is too much to remember and it is hard to learn to use it.

In general, large tagsets seem to be considered harder to manage that smaller ones, and they seem to reduce the accuracy of the annotation. Interestingly enough, however, Sánchez León & Nieto Serrano (1997:155-156,164) found that with a certain kind of probabilistic model, a very large tagset (475 linguistic POS tags) resulted in better accuracy than a small tagset. They did not tag English, however, but Spanish, but the result is still worth noting. They also mention that it has been shown by Elworthy (1995) that larger tagsets can obtain even better accuracy than smaller ones.

A slightly different problem, related to this, is that it is difficult to decide which category some words belong to in the first place. That is, in addition to the problems of deciding which category of several possibilities a word belongs to, sometimes it is hard to say what the possible categories are. It is easy to imagine that this becomes a problem especially with large tagsets, where very fine distinctions should be made. This is also a problem that should be kept in mind when a tagset is designed. (cf. Greenbaum 1993:14-15.)

In order to determine how many and what kinds of tags there should be in a tagset, one must decide how much time and money one is willing to put into the process of tagging. Some types of tags are easy for the computer to add, but even human annotators have problems coping with some of the others. The point here really is that it is never possible to tag every possible detail, and

it must be decided what kind of word-categories are actually needed. The rest have to be left out.

3.5.2 Problems with part-of-speech tagging

There are certain problems that come up with grammatical tagging. Most of them have to do with the fact that there are idiomatic sequences of words and other word combinations, which make it hard to assign tags to individual words. There are also problems with ambiguous words, which could be assigned to more than one word class. In addition, no matter how carefully the tags are assigned, there always is some degree of error with the tags. This section will consider the problems of tag assignment and possible solutions to them.

Leech (1997b:21-24) recognizes three different kinds of problematic cases in assigning tags to words. These are **multiwords**, **mergers** and **compounds**. These are all cases where one orthographic unit (a word surrounded by spaces) does not correspond to one morphosyntactic unit (the unit that has to be identified for tagging). The following sections will explain each of these in detail. After that, sections 3.5.2.4 and 3.5.2.5 deal with ambiguous words and error rates.

3.5.2.1 Multiwords

Multiwords include expressions like *in spite of* and *in lieu of.* McEnery & Wilson (1996:40) refer to these as "idiomatic sequences of words" and give as an example *so that.* In all of these examples, although there is more than one word, the words together function as a single unit. In some cases, there may be problems in defining which expressions are multiwords and which are not: for example, Leech (1997b:21-22) notes that *provided that* can be treated as a multiword or not.

One solution for dealing with multiwords is to give them so-called **ditto tags**. Ditto tags include the normal grammatical tag, and in addition to that, two numbers: first, a number denoting how many words belong to the

unit, and second, a number denoting which part of the unit the word is. For example, *in spite of* could be tagged as follows:

Example 12: Multiword tagged with ditto tags

In_PREP31 spite_PREP32 of_PREP33

(Leech 1997b:21)

In this example, the tag PREP tells that the expression is a preposition. Number 3 in each tag tells that the unit consists of three words, and the number after that tells which part of the unit is in question (*in* is the first, *spite* the second and *of* the third part).

In this case, the problem could also be dealt with TEI tags. As Leech (1997b:31) presents it, a similar expression, *in lieu of*, could be tagged also in the following way:

Example 13: Multiword tagged with TEI tags

<w PRP>in lieu of <w NN1> payment

(Leech 1997b:31)

The first tag <w PRP> (preposition) refers to all the words that follow it before the next tag occurs (in this case <w NN1>, noun, which refers to *payment*).

However, some multiwords may be discontinuous (e.g. Leech 1997b:21). For example, in phrasal verbs there may be something between the different parts of a unit (e.g. *give it up*). It should be possible to deal with these with ditto tags, but applying such tags may prove problematic. For instance, in case of *give it up*, there could be a longer noun phrase between *give* and *up*. In that case, the computer should be able to recognize the syntactic unit NP in order to make the connection between *give* and *up* (cf. Garside & Rayson 1997:189).

3.5.2.2 Mergers

The second type of problematic words are mergers. They consist of one orthographic unit which corresponds to more than one morphosyntactic unit

(Leech 1997b:22). Examples of this include words like *don't*, *can't*, *shan't* and *dunno*. A "word" like *can't* really consists of an auxiliary verb and a negative, which it would be useful to tag separately to make searching the corpus easier.

The problem with these is that for tagging purposes, it may not be possible to divide the word into parts that still make sense (Leech 1997b:22). The division of a word like *don't* will produce understandable units like *do* and *n't,* where *n't* can be recognized as *not*, but the breaking down of *shan't* will result into the weird unit *sha* which does not look very natural. A word like *dunno* is even harder, since the original words (*do* + *not* + *know*) are not distinguishable anymore (cf. Leech 1997b).

There are no generally agreed-upon methods to annotate these, but Leech (1997b) presents a way to handle them. The relevant tags can be simply inserted in the middle of the words, for example

Example 14: Mergers (1)

Sha_VAUXn't_NEG

(Leech 1997b:22)

or with TEI tags,

Example 15: Mergers (2)

<w PRP>they<w VBB>'re

(Leech 1997b:31).

McEnery & Wilson (1996:41), in turn, suggest that there could be more than one tag assigned to one word, so that *don't* might be tagged

Example 16: Mergers (3)

Don't_VD0+XX

(McEnery & Wilson 1996:41)

where the "+" connects the two tags. "XX" here is the tag that CLAWS uses for *not*, and "VD0" is a special tag for *do* (Garside, Leech & McEnery (eds.) 1997:257).

However, neither of these methods of tag insertion is without its faults, and one should consider carefully whether to use them or try to find another way to deal with mergers. One way is to expand the contraction into the full words that it consists of, but this has its drawbacks, too, in that the form of the word in the original text is lost (Leech 1997b). It is very hard to try to come up with an annotation scheme that could deal with all the possible cases.

McEnery & Wilson (1996:40-41) consider the matter of mergers the other way around, and talk about recoverability: if an expression like *don't* has been expanded for annotation purposes into *do not*, the user should still be able to get the original form *don't*. It is useful here to remember the seven maxims of annotation that were explained in section 3.1.3. The first of them was that it should always be possible to remove the tags and restore the text to its original form. Keeping this in mind, mergers should be somehow tagged in a way that would make the annotation sensible, but also retain the original forms. In CLAWS, this problem was solved by expanding the words for tagging, but inserting additional markup to show that there originally was a contracted form (McEnery & Wilson 1996:41).

3.5.2.3 Compounds

The third type of problematic cases are compounds (Leech 1997b:22-24). These are units in which one or more orthographic units correspond to one or more morphosyntactic units (Leech 1997b:22). Some words, like *eyestrain*, may be spelled in three different ways: *eye strain, eyestrain* or *eye-strain*. Which one of these is used depends on the writer's view of whether the word is a compound word or two separate words. All of these could be tagged either differently or in the same way: often, corpus compilers tend to take the easy way and tag them according to their orthographic form. However, as Leech (1997b:23) points out, this is not the best possible solution to the user. If all the forms (*eye strain, eyestrain, eye-strain*) are tagged differently, three different searches have to be made to retrieve them all.

Automatic tagging can also result into strange words sometimes. Leech (1997b:23-24) gives two interesting examples: a "compound word" *York-San* from the phrase *New York-San Francisco flights*, and *post-Cold* from

post-Cold War attitudes. An automatic tagger really cannot tell these apart from normal compounds with a hyphen. In order to tag these correctly, the tagger should be able to deal with e.g. *New York* and *San Francisco* as multiwords, and see the phrase *New York-San Francisco* as a larger compound (Leech 1997b:24).

These kinds of compounds can, however, be dealt with by TEI tags. For example, *post-Cold War* could be tagged as follows:

Example 17: Compounds with TEI tags

```
<w AJ0>
<w PRP>post-</w PRP>
<w AJ0>Cold</w AJ0>
<w NN1>War</w NN1>
</w AJ0>
```

(Leech 1997b:31)

The point to be made here is that the first and last tag (AJ0, adjective) surround the whole unit and do not leave any doubt as to which words belong to the compound and which do not.

3.5.2.4 Ambiguous words

Yet another point to consider are words that cannot, even in their context, be unambiguously assigned a word class at all. These would include examples like *gold watch* and *plastic bottle* (Leech 1997b:32) – are *gold* and *plastic* in these phrases nouns or adjectives? Other examples, also given by Leech, would include cases like *The Pope, Auntie, Times Square, Fifth Avenue* and others, in which there may be dispute over whether the phrase or a part of it is actually a proper noun or a common noun (e.g. *Times* in *Times Square*). One solution would be to apply a single tag to the whole phrase (*Times Square*) and thus treat it as a multiword (Leech 1997b:32).

3.5.2.5 Error rates

One more thing to mention here is that all automatic taggers leave a certain amount of errors. McEnery & Wilson (1996:126) say that error rates are

nowadays usually around 4-5%. Smith (1997:147) notes that even after running several programs that tried to correct errors in the British National Corpus, there was about 2% error rate left.

The point here is that human language is so complex that no matter how many programs one tries to develop to deal with it, there always will be some degree of errors left. Because of the natural ambiguity of language not even human annotators can deal with all the problems that may be encountered.

In the preceding sections I have discussed problems associated with POS tagging. Problems are caused especially by expressions where one orthographic word does not make up exactly one unit that should be tagged. In some cases, several words form one unit (such as *in spite of*) and sometimes, conversely, there is only one orthographic word that consists of several units that should be tagged separately (such as *dunno*). There are also occasionally different ways to spell a word, so that it sometimes looks like one word and sometimes like two or more words, which makes tagging harder. In addition, there are ambiguous words that require difficult tagging decisions even from human annotators. No doubt ~~that~~ one may encounter even more problems, in addition to the ones mentioned here. In practice, there are likely to be problems relevant to a specific corpus that cannot be predicted beforehand.

Before moving on to another type of annotation, there is one more point that has to be considered: the grammatical tagging of languages other than English.

3.5.3 Part-of-speech annotation for other languages

Most taggers have been developed for the English language. However, there have been attempts to adapt some taggers for other languages, and indeed, one of the features of a modern tagger could be that it can process more than one language (Smith 1997:138). Leech (1997a:9) notes that language-independent taggers have been developed, and McEnery & Wilson (1996:125-126), too, mention a tagger that is able to do several languages. They call it the Cutting tagger, which presumably is the same as the Xerox tagger mentioned in other studies.

In fact, some taggers are at least theoretically suitable for any language, since they can be trained with any previously constructed corpus: they can pick up the lexicon and probabilistic model from any POS-tagged corpora that is given to them and use the information thus gained to tag further corpora. In other words, such taggers can by themselves, without human intervention, find out the rules for any language, as long as a tagged corpus for that language is already available.

However, Sánches León & Nieto Serrano (1997) tested a supposedly language independent tagger, the Xerox tagger, for Spanish, and found out that there were definite deficiencies in it. For example, it tags the words according to the word-form, but does not do any morphological analysis. For a language like Finnish, which is highly inflected and has over a dozen cases for words, such would be necessary.

The Xerox tagger only selects the most probable tag for each word. However, as Sánches León & Nieto Serrano (1997:153) note, in case of highly inflected languages there often is a clear correspondence between "(linguistically motivated) suffixes and morphosyntactic properties of the word(s) they are attached to". In other words, the tagger should not try to rely on probabilities in cases where it is known that a word with a particular suffix necessarily belongs to a certain word class. A tagger should be able to take advantage of this kind of knowledge, or in other words, "make use of a morphological component" (Sánches León & Nieto Serrano 1997:153, 162).

Another problem with the Xerox tagger was with what Sánches León & Nieto Serrano (1997:161-162) call tokenization. The tagger could not recognize fixed phrases or other expressions that consist of more than one word as one unit. This seems to have been a rather big problem in case of Spanish, and the researchers suspect that it could be even worse with some other languages.

It should be remembered that despite its claimed language-independence the Xerox tagger was originally developed and tested for the English language. There are taggers that were originally meant for some other language (e.g. Finnish), and no doubt work rather well for that specific language. However, as was noted in the beginning of this section, the aim

nowadays is to develop tools that are language-independent, or at least easily adaptable to other languages. Such tools would ease the tasks of many researchers, make both corpora and corpus software such that they could be more easily reused or combined, and also help the standardization of both annotation and work methods.

In this chapter I have examined part-of-speech annotation, how to add it to a corpus and what kinds of problems can be expected with it. The beginning of the chapter described part-of-speech annotation and its possible uses. Section 3.5.1 explained how POS tags can be added to a corpus, and how different types of automatic taggers work. It also gave some advice about tagsets. Section 3.5.2 listed different types of problems that are common in POS tagging, and the final section (3.5.3) examined annotating languages other than English. The following chapter will move on to the next most common type of annotation: syntactic annotation, more commonly known as parsing.

3.6 Parsing (syntactic annotation)

This chapter is about the parsing of corpora. **Parsing** means that the sentences in a corpus are analysed syntactically. Sentences can be divided, for example, into constituents like noun phrase, verb phrase, and so forth. In other words, parsing is about producing phrase structure trees for each sentence. Parsing is also known as **syntactic annotation**, and corpora containing such annotation are often referred to as **treebanks**, since they consist of structures that could be represented as syntactic trees.

The following is an example of a parsed sentence from the British National Corpus.

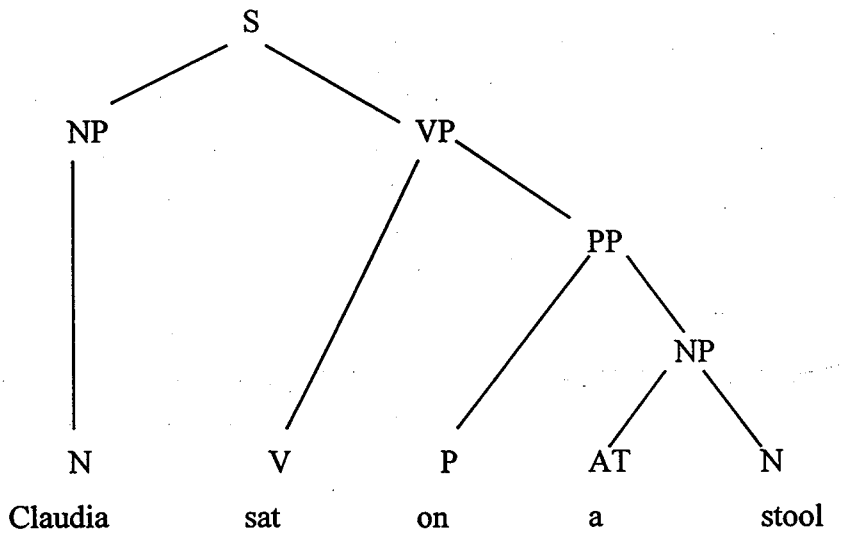
Example 18: Parsed sentence

[S[NP Claudia_NP1 NP][VP sat_VVD [PP on_II [NP a_AT1 stool_NN1 NP] PP] VP] S]

(McEnery & Wilson 1996:43-44)

Familiar elements of syntactic analysis are easy to find in the example. The brackets single out each constituent, which are also named (eg. NP=noun phrase, VP=verb phrase). Note that the sentence also has part-of-speech tags. The same example sentence could also be represented as a syntactic tree:

Example 19: Syntactic tree



(adapted from McEnery & Wilson 1996:43)

Parsing can be seen as a natural continuation of part-of-speech (POS) annotation. After POS tagging has been performed for each word, it is possible to annotate higher-level syntactic relationships in the text (McEnery & Wilson 1996:42). Leech (1997b:19) points out that there are strong arguments for claiming that these two levels of annotation, POS and parsing, are not distinct levels at all, but that POS tagging only specifies the "leaves" of a phrase structure tree. Greenbaum (1993:12-13), too, notes that the decisions the TOSCA corpus project made about their POS tagset were influenced by the fact that they were going to parse it later on. Parsing is, indeed, the second most common type of annotation, after POS tagging (McEnery & Wilson 1996:43).

Leech and Eyes (1997:41-48) and Bateman et al. (1997:168-169) mention a few well-known treebanks. These include, for example, the Penn treebank (3,300,000 words), Nijmegen treebanks: Nijmegen Corpus (130,000) and TOSCA corpus (1,000,000), The Suzanne Corpus (128,000) and the Helsinki Constraint Grammar Corpus. All of these corpora are parsed slightly differently, and that is indeed one characteristic of parsed corpora: there is no established standard practice for how syntactic annotation should be done, although certain general guidelines can be given (Leech & Eyes 1997:36, Kahrel et al 1997:237). Especially when compared to POS tagging, the field of syntactic annotation is still developing: there are more ways for doing it, but the accuracy is lower (McEnery & Wilson 1996:49).

The reasons for adding syntacting annotation to corpora are much the same as for POS annotation. It makes the use of the corpus easier and enables more powerful searches. However, there are some additional reasons why parsing a corpus is useful.

A parsed corpus can bring out idiosyncracies in language that have not been noted before (Bateman et al. 1997:167). Assigning a correct syntactic structure for any given sentence is not an easy task, the problem starting with how we define "correct". Parsing may help to bring out the kind of problems there are in syntactic analysis once real-life sentences are taken to be analysed.

As all corpora, parsed corpora, too, can challenge people's intuitions about language. Sampson (1991:184) gives an interesting example of this. Textbooks of linguistic theory tend to present two sentence types as being the most typical sentence structures in English. These are the subject-transitive verb-object type (*John hit Mary*) and the subject-intransitive verb type (*Mary wept*). A corpus search, however, showed quite different results. The first type mentioned was the most common one, but the second, as in *Mary wept*, was extremely rare. The second most common type of sentence consisted of a single noun phrase with no verb, which can be frequently found in headings and captions. This is a good example of the uses of parsed corpora, since it proves something about language that could not be noticed without syntactic annotation.

Parsed corpora are also very important in natural language processing. Patten (1992:29) notes that first parsers were developed already in the 1950s, in order to produce machine translation systems. They did not produce the required results since, of course, translation is about much more than just translating individual words and structures. Many systems that help human translators, however, were developed. Patten (1992:29) also mentions that parsing is important in developing systems that allow people to interact with computers in a natural language such as English, instead of using computer languages.

Parsed corpora can also be used to compile computer lexicons. In addition to containing just the words themselves, such lexicons can have information of the words' frequencies in the corpus, as well as frequencies of their collocations, lemmas, etc. (Leech & Eyes 1997:35-36). Such lexicons, apart from any traditional linguistic use they may have, can be of great importance in computational linguistics. Leech & Eyes (1997:34) also point out that for speech recognition and machine translation ever to be successful, in addition to there being a good lexicon available, one must be able to analyse sentences into their constituents and their relations, and this is exactly what parsing provides.

In addition, there are more and more electronic texts available all the time, and their uses are many. Patten (1992:30) notes that simple keyword

searches yield only about 20% of the relevant information, and that parsing could help to index all the information automatically. A rather interesting application of parsing involves "skimming a data source for information on a particular topic" (Patten 1992:30). Such systems find many uses in political and industrial intelligence gathering. Parsing systems can also be useful in computer-aided instruction and in the automatic proofreading that word-processors do (Patten 1992:30).

One use for parsed corpora that frequently comes up is the development of more advanced parsers, that is, programs that automatically add syntactic annotation to corpora (eg. Leech & Eyes 1997:34-35). Automatic parsing and parsers can be developed and tested by taking advantage of information from already existing corpora. For example, probabilities for the likelihood of certain phrase structures occuring can be deduced from previously parsed corpora. This is more or less the same that can be done with POS taggers; old corpora are used to train new software.

Parsing, as well as POS tagging, is also more or less a requirement for some higher level annotations. For example, in order to include anaphoric annotation to a corpus, parsing is necessary (Garside & Rayson 1997:180).

Syntactic annotation can be added either manually or automatically. The following section describes how parsing can be done and what is required for it.

3.6.1 How to add syntactic annotation

Practically all automatic parsing systems nowadays require at least some amount of human intervention. The amount of human editing may range from almost completely manual analysis and input, where the computer is only helping the editing, to a system where the computer does the job and humans check it afterwards (Bateman et al. 1997:167).

McEnery & Wilson (1996:129-130) give a list of the operations that an automated parsing system should be able to do. First of all, it should be able to indentify the words of a sentence and assign syntactic description to them. After that, it should group the words into units such as phrases, which form the main syntactic constituents of the sentence. In addition, it should be able to

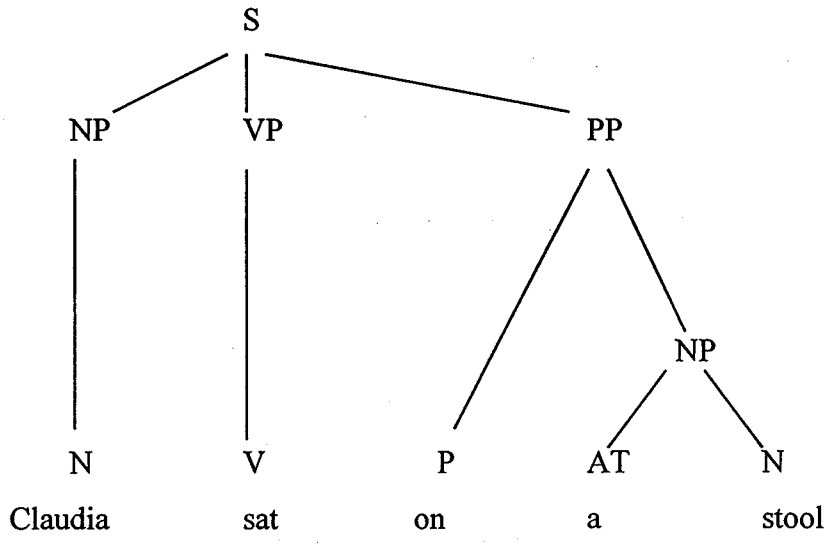
name the constituents correctly. McEnery & Wilson (1996:129-130) also note that the parser should do this with a high degree of accuracy for any given sentence, and Bateman et al. (1997:166) note the same: a parser should be able to analyse "any sentence of naturally occuring *unrestricted* English". How many parsers actually get anywhere close to this requirement is a problem that is dealt with in more detail in section 3.6.3 of this chapter. Suffice it to say here that most parsers perform well with invented example sentences, but run into serious problems with instances of real language.

The method presented by McEnery & Wilson seems to be what Patten (1992:31) calls **bottom-up parsing**. It means that the parser starts with the smallest units and tries to build up to a whole sentence. The other possibility is **top-down parsing**, which means that the parser starts with a whole sentence and tries to chop it down to smaller pieces (Patten 1992:31.35). This will not be discussed in more detail here; there are far too many different types of parsing to try to explain them all at once, and it would not serve the purposes of the present paper. However, Patten (1992:31) makes a rather interesting comment about parsing which is worth quoting here: "The parser can easily determine the topmost and the bottommost levels of the tree in advance: the trick is to fill the gap between". This, indeed, applies to any type of parsing.

Just as a POS tagger needs a tagset and a tagging scheme, a parser needs a **parsing scheme**. It should include a list of all the symbols that may be used in the annotation, a basic description of the symbols and instructions for how the symbols are to be used (Leech & Eyes 1997:37). Such a scheme is likely to improve and develop all through the annotation process, and the end result should provide the users with a manual explaining what kinds of tags they can expect to find in the corpus. It may also be a good idea to give the users example phrases from the corpus so that they can see in practice how it has been annotated (Leech & Eyes 1997:37).

McEnery & Wilson (1996:44) point out that the tagging schemes used by different annotation projects may vary greatly: both the number of constituent types and the way they are applied may be different. For example, the sentence *Claudia sat on a stool* could be parsed as in example 19, but an alternative way would be to divide it in the following way:

Example 20: Alternative syntactic tree



(adapted from McEnery & Wilson 1996:45)

There seem to be many different views of grammar that can be used in parsing. A couple of them will be presented in section 3.6.2.2. It should be remembered, however, that the choice of the parsing scheme should be governed by the uses of the corpus rather than by any ideological preferences of what is the "correct" way to present a sentence structure.

The main reason for needing a parsing scheme is to keep the annotations accurate and consistent (Leech & Eyes 1997:39-40). It has to be defined what is a correct parse and what is not: even human annotators may have different opinions of them. In order to maintain consistency in the corpus, so that each similar sentence is tagged in the same way, rather strict guidelines have to be established. How strict, exactly, is another question then: Leech & Eyes (1997:37-38) note that for human annotators there does not have to be a comprehensive grammar from the beginning, but rather a set of guidelines that can be developed as need arises. Too exact a parsing scheme also has the disadvantage that it will be too hard for human annotators to remember and use. Marcus et al. (1993:283) point out that it takes much longer for human annotators to learn syntactic annotation than POS tagging in any case; this even though, in their study, they used a rather simple parsing scheme. Therefore, it would not be practical to have a very complex set of rules from the beginning.

Parsing is not nearly as successful as POS tagging. Depending on the source of information, many different success rates have been quoted. These range from 30-40% (McEnery & Wilson 1996:130) through 70-80% (Leech & Eyes 1997:35) to even as high numbers as 94-96% (McEnery & Wilson 1996:134). The differences are mainly due to the differences in the way the parsers work. These will be dealt with in the following section.

3.6.2 How an automatic parser works

There are many ways to classify parsers. On one hand, they can be classified by how the parser actually assigns the appropriate tags to the constituents; on the other, there are many theories about how sentences are divided into smaller constituents. The following sections will give a couple of examples of these and try to explain them. Note that there are numerous ways to parse sentences, and the ones presented here do not cover the whole field of syntactic annotation. These represent, however, some of the general principles that seem to crop up quite often.

3.6.2.1 Rule-based, probabilistic, hybrid parsers

As with POS taggers, there are rule-based, probabilistic and hybrid parsers. Rule-based taggers have a set of rules about sentence structure that they apply (McEnery & Wilson 1996:131), whereas probabilistic taggers use probabilities derived from earlier corpora to decide which is likely to be the correct parse for a sentence (Leech & Eyes 1997:35). Hybrid systems can use a combination of human-invented rules and corpus statistics to come to decisions about the correct parses (McEnery & Wilson 1996:133). The best results are usually achieved by probabilistic and hybrid systems (Bateman et al. 1997:166, McEnery & Wilson 1996:134). It has also been noted that the best results are achieved by parsers that use previous corpora in the first place – that is, parsers that have been trained with existing corpora - rather than by traditional parsers that are more or less based on human-invented rules (McEnery & Wilson 1996:135). There are also opposite results, however: for example Bateman et

al. (197:166) note that systems that do not use statistics can also work very well.

3.6.2.2 Phrase structure and dependency grammar

There are two underlying theories that are mostly used for modelling syntactic structures on computers. They are phrase structure grammar and dependency grammar (Kahrel et al. 1997:239, McEnery & Wilson 44-46). Phrase structure grammar uses constituents such as noun phrase, verb phrase, adjective phrase etc, and marks the hierarchies in a sentence. Dependency structure, on the other hand, tries to identify units such as adverbial, determiner, premodifier, subject and object, that is, the interdependencies between the words of a sentence (McEnery & Wilson 1996:46-49).

There are also other types of theories, some of which have nothing to do with linguistic views of sentence structure (McEnery & Wilson 1996:132-133). Such systems go through large parsed corpora and form their own ideas of what are likely to be typical structures in language. In other words, they collect the probabilities for different kinds of structures that they can find in the text, and apply them when they have to select the correct parse from several possibilities.

Yet one more difference that can be made between different parsers is that of **full parsing** and **skeleton parsing**. Full parsing means a parsing scheme that is as detailed as possible, whereas skeleton parsing uses fewer tags (McEnery & Wilson 1996:44). Skeleton parsing was originally developed for human annotators, for whom a larger tagset would have been problematic. Not only does skeleton parsing make manual parsing faster, but it also reduces inconsistencies and inaccuracies (Leech & Eyes 1997:36-37). For example Marcus et al. (1993) used a skeletal parsing scheme for the parsing of the Penn Treebank. Their tagset, in fact, had only 14 actual syntactic tags and a few other special elements (Marcus et al. 1993:281).

In addition to the parsing techniques presented here, there are also many others. For example Patten (1992:35-43) introduces several different types of parsing methods, some of them seemingly rather old. It must be

remembered that automatic parsing has been done for a long time (since the 1950s) and there have been many methods over time to do it. All of them have their advantages and disadvantages. Since new methods are still being developed all the time, it seems that no single satisfactory method has yet been found.

One more parsing system that is worth mentioning here is the ENGCG system that has been developed in Finland, at the University of Helsinki. The parser uses a constraint grammar model and consists of two parts, a morphological analyser and a reductionistic parser (Voutilainen & Heikkilä 1994:189). The authors also say that the system can be adapted to languages other than English (Voutilainen & Heikkilä 1994:191). More information about ENGCG, as well as an on-line demo, can be found at http://www.conexor.fi .

3.6.3 Problems with parsing

The problems with parsing seem to center around one theme: how to define the rules for parsing, and how to apply the rules.

First of all, it is hard to come up with sufficient rules in the first place. As McEnery & Wilson (1996:131) point out, it is not a good idea to try to sit down and write rules that would make possible the parsing of any given sentence. It is possible to make a working set of rules for a strictly defined subdomain of language, but for larger amounts of more varied data, the same rules will simply not work anymore. This is the problem of "scaling up": if only 100 grammar rules and 1000 words were allowed in a language, parsers would work fine, but when we move on to the real world where basically an infinite amount of both is allowed, the results are less encouraging (McEnery & Wilson 1996:136).

Leech & Eyes (1997:37) also make the point that in practice, the need for new rules does not seem to disappear at any point when new data is processed. One might expect that at some point, after enough data, one would have all the required rules; however, this does not seem to be the case. It is clear that 100,000 words need more grammar rules than 1000 words, but it seems that even when 1,000,000 words, or several million, have been parsed, new rules just keep coming up.

Both Leech & Eyes (1997:37) and Bateman et al. (1997:170-171) note that at some point, if human annotators are used, the amount of rules becomes too large. It is difficult for the annotators to keep track of all the changes, and previously parsed sentences should be updated to conform to the new rules. Not only does this make the annotation process much slower, but it also adds to inconsistencies. Bateman et al. (1997:176) note that a huge set of rules also has a drawback for automatic parsers: since there are more possible rules available, the number of incorrect parses will increase. There is, indeed, the problem of maintaining both quantity and quality: how to keep a parser (human or automatic) productive and able to parse many different constructs, and at the same time, maintain accuracy. A parser should be able to account for all structures of ordinary language but also define its grammar so that incorrect parses would not be generated (Bateman et al. 1997:167-168).

Another aspect of language that creates problems is that natural language is quite often ambiguous, and several interpretations of the structure of a sentence are possible. Such cases, in case of manual parsing, create inconsistencies, especially when more than one human annotator is involved and no strict rules exist (McEnery & Wilson 1996:49). In the case of automatic parsing, the computer is likely to have many possible parses for the given sentence and cannot decide which one is correct.

As an example here, Leech & Eyes (1997:41) note that 12-15% of prepositional phrase constructs, which are very hard for automatic parsers, cannot be annotated consistently even by human annotators. This is true even though humans know about the real world, and can look at the meaning and context of the phrase. McEnery and Wilson (1996:133-134), too, note that sentences can often have many potential parses, and give an example of a sentence with a prepositional phrase, *the dog heard the man in the shed.* This sentence could be interpreted in several different ways: who was in the shed? What does the prepositional phrase *in the shed* modify?

Some of these ambiguities cannot be resolved even by human annotators. In such cases, one just needs to establish a rule of what is to be done and force a certain interpretation on the sentence, even though it is not the only correct one. In automatic parsing, using probabilities may help to solve

ambiguities. By taking advantage of the frequencies of how similar sentences were parsed earlier, a parser can decide which one of the possible parses is the likeliest to be the correct one (McEnery & Wilson 1996:134).

A whole new area of problems comes up if one wants to parse spoken language. In addition to having at least as many ambiguities and context-dependent sentences as written language, there are many new problems. Leech & Eyes (1997:49) point out some of these, and note that very little has been done in this field so far. The annotation scheme for spoken language should take into account characteristics of speech such as incomplete utterances, unplanned repetitions and false starts, and it is necessary to devise new symbols to represent these.

In this chapter I have discussed parsing, which is also known as syntactic annotation. In the beginning of this chapter I defined what syntactic annotation is and what its uses are. Section 3.6.1 described how to add syntactic annotation, and section 3.6.2 explained how an automatic parser works. The last section (3.6.3) looked at the problems that may come up with syntactic annotation. The following chapter will move on to the next higher level of annotation, which is that of semantic annotation.

3.7 Semantic annotation

This chapter will deal with a type of annotation that is not nearly as common as the two previous ones: **semantic annotation**. Theoretically, there might be many types of annotation that could be called semantic annotation. For example, one could try to identify the agents, instruments and other such items in a text and mark the semantic relationships between them (McEnery & Wilson 1996:49). In practice, however, this kind of analysis has not been done to any great extent. There is only one type of semantic annotation that has been used more than tentatively: annotating the word senses, or semantic features, of the words in a text (McEnery & Wilson 1996:50). Other types of semantic annotation have been attempted, too, but there are not many experiences with them yet. The following is an example of semantic annotation.

Example 21: Semantic annotation

And	00000000
the	00000000
soldiers	23241000
platted	21072000
a	00000000
crown	21110400
of	00000000
thorns	13010000
and	00000000
put	21072000
it	00000000
on	00000000
his	00000000
head	21030000
and	00000000
they	00000000
put	21072000
on	00000000
him	00000000
a	00000000
purple	31241100
robe	21110321

The numeric codes stand for:

00000000	Low content word (and, the, a, of, on, his, they etc)
13010000	Plant life in general
21030000	Body and body parts
21072000	Object-oriented physical activity (e.g. put)
21110321	Men's clothing: outer clothing
21110400	Headgear
23231000	War and conflict: general
31241100	Colour

(McEnery & Wilson 1996:51)

This example uses categories like, for example, "plant life in general" and "colour". Note that the codes for the categories here form hierarchies: everything starting with 2, for example, belongs to the top-level category "Man" (McEnery & Wilson 1996:50).

There is, however, no agreement on what kinds of features should be annotated. The earliest semantic annotations (in the late 1960s) were not done with linguistic methods at all, but with social science definitions to examine matters like social interaction or political behaviour (McEnery & Wilson 1996:50). There are some requirements for how the categories should be defined, and they will be dealt with in the following section. Before that, however, there will be a few words about the uses of semantically annotated corpora.

Wilson & Thomas (1997:64-65) give some examples of the areas in which semantic annotation can be useful. One of them is, understandably, semantics, but discourse analysis and content analysis can also take advantage of it. Such annotation can also prove necessary in computational linguistics, in both message understanding and language generation, and in other tasks where the computer has to understand word senses.

In addition, for any corpus user, semantic annotation can prove very useful because it can help information retrieval. Wilson & Thomas (1997:53-54) give an example of this, too. When searching from a corpus, one should be able to identify related words. For example, if you want to find something that has to do with clothing, it is not enough to search for *trousers*. This is because the texts might just as well refer to them as *slacks, shorts, leggings, pants*, or whatever. In a case like this, semantic annotation can prove to be very useful. With it, there is no need for the user to try to think of all the possible synonyms for *trousers*. Since the words in the corpus have been divided into semantic categories, the search can be made for the relevant category instead of individual words.

The same works the other way around: one may need to identify the senses of words that look the same, so that only the relevant instances are retrieved. For example, the word *boot* could be used to mean footwear, but it has many other senses, too, for example *to boot a computer, the boot of a car,*

to boot something (=kick) and *licking a person's boots*. Semantic annotation helps to retrieve only those instances of *boot* that are wanted, not all of them.

A more practical use for semantic annotation is presented by Wilson & Rayson (1993). They used semantic annotation to tag transcribed spoken interviews between market researchers and members of the public. Normally, such market surveys are based either on questionnaires, which offer a limited number of choices, or on in-depth interviews which tend to produce small samples and no reliable statistical data (Wilson & Rayson 1993:215-216). With semantic tagging, however, it is possible to perform automatic content analysis on large amounts of data.

3.7.1 How to add semantic annotation

Semantic annotation can be added manually, with computer assistance or (theoretically) fully automatically (Wilson & Thomas 1997:62). Computer-assisted annotation makes use of a lexicon, and assigns possible tags to all words. Foreign, otherwise unknown words and words for which the computer cannot select the correct tag from all the alternatives are left to human annotators. Garside and Rayson (1997:188) note that so far there are no fully automatic, accurate semantic taggers available. Even those that are under development are such as described in the beginning of this chapter, and tag only individual words. In the long run, it would be useful to have semantic taggers that would understand the meaning of sentences or other larger units. This does not yet seem possible today, however.

In any case, there are semantic taggers that perform their task automatically to a certain point. For example Wilson & Rayson (1993:218-225) present an automatic tagger for semantic annotation. The program uses a lexicon and part-of-speech tags to give each word in a text a relevant semantic tag. The program can also apply morphological rules to remove word endings and it can revise the word-class information to reflect the word-class of the stem of the word. The program also recognizes some idioms and phrasal verbs (Wilson & Rayson 1993:221-222). At least at the point of development that was reported, though, the annotated texts still needed human beings to select

the correct tag in many cases. The program left about 20% of the words with more than one semantic tag (Wilson & Rayson 1993:222-223).

Garside & Rayson (1997:189) also note that both POS tagging and parsing are useful as a basis for successful semantic annotation. POS tagging can be seen as a requirement, and parsing, too, can help to discriminate word senses. It may be noted that POS tagging already makes some distinctions in meaning explicit by naming the word classes for the words. For example, the difference between *book* as a noun and a verb has been made clear at that point.

Patten (1992:47-49), in fact, turns the matter upside down and suggests that semantic criteria could be used to guide a parser. For example, the system could recognize that the word *give* is found in a sentence, and that it is the main verb. After that, semantic knowledge would tell the parser that such roles as ACTOR (the giver), OBJECT (the thing being given) and TO (the receiver) must also be filled (Patten 1992:47). The system would also know what kinds of words can be ACTORs, for example. Thus, semantic information would guide the parser to recognize the rest of the units in the sentence.

Garside and Rayson (1997:190-192) also list a few other points that a semantic tagger could take into account. These will be mentioned here only briefly. A semantic tagger could take advantage of probabilities of words belonging to certain categories, just like POS taggers and parsers. In addition to that, however, it could use what it knows about the domain of discourse and other parts of the same text. For example, in a financial domain, *bank* is more likely to mean a financial institution than anything else. Wilson and Rayson (1993:222), though, found that such information did not help very much in practice – the disambiguation ratio was only 3% better, since there were still a large number of ambiguous words that did not belong to a specific domain of discourse.

Garside & Rayson (1997:191) also say that it is claimed that the meaning of words tends to stay the same throughout a text: if earlier in the same text *bank* meant "side of a river", it is likely to mean that again when in occurs repeatedly in the same text. The immediate context (preceding and following words) can also be of use: if *bank* is surrounded by other words that have to do with finances (eg. *account, money*), *bank* is likely to belong to the

same category. For example Guthrie (1993:232-234) presents a method that uses "neighbourhoods" of words to disambiguate word senses. Janssen (1992:143), too, notes that "because certain words typically co-occur [...], they reinforce (or aid) each other's interpretation". She gives the following example: *This tennis tournament is played on a clay court*, where the word *tennis* makes the 'sport' interpretation of *court* more likely. In addition to these requirements, the tagger should also be able to deal with idiomatic expressions.

The technical side of semantic annotation aside, there is the matter of defining an annotation scheme and the semantic categories that may be used. There are words that are "connected in some way with the same sphere of activity" (Wilson & Thomas 1997:54), although they are not synonyms, hyponyms, or in any other such relation to each other. Wilson & Thomas (1997:54) give the example that the field of equestrianism could include words like *rider, horse, eventing, spurs, saddle, dressage,* and *jump-off.* These words could clearly belong to the same category, because they obviously have something in common. Note that these words do not refer to the same thing, nor are they opposed to each other; rather, they have something to do with the same field of activity.

Wilson & Thomas (1997:55-57) list criteria for defining semantic categories. First of all, the categories should make sense in linguistic or psycholinguistic terms. Secondly, the semantic fields should cover all of the vocabulary in the corpus. The fields should be flexible so that they can be extented to handle texts from different periods, languages and registers, and they should work at an appropriate level of granularity. The last point means that the fields should be arranged hierarchically, and that they should not be either too specific or too general. For example, there could be a hierarchy like "animal" → "mammal" → "monkey", in which "animal" is the highest, most general term, and "monkey" the most specific. If the categories are either too general or too specific, they do not help the user very much. Lastly, the annotation scheme should conform to a standard, if one exists. As yet, there is none, but such a standard could, at least, define some basic high-level categories and thus help to make semantically tagged corpora more compatible with each other.

In practice, as Wilson & Thomas (1997:54-55) point out, semantic annotation schemes have to compromise between how words are classified in the human mind, and how they can be classified by computers. There is the problem that is common in corpus annotation: what is wanted and what can be done are not the same thing. The following section will examine problems of semantic annotation.

3.7.2 Problems with semantic annotation

This section briefly examines some of the problems in semantic annotation. First of all, there are a couple of problems that have to do with the purely technical side of semantic annotation. Wilson & Thomas (1997:62) point out two reasons why semantic annotation is so much harder to automate than, for example, POS tagging.

The first reason is that semantic annotation is inherently knowledge-based. Human annotators know the meanings of words because of what they know about the real world, but a computer would require a lexicon that contains all the different words in the corpus. The second reason is that it is not known yet whether the same kind of statistical calculations that are used for determining probabilities in POS tagging can be used in semantic annotation. POS taggers can "guess" the word-class of an unknown word from the surrounding words; however, it may not be possible to do the same with semantic categories. (Wilson & Thomas 1997:62)

Guthrie (1993:227) also notes that even humans do not always agree about what a word means in a particular sentence. As she points out, not even lexicographers always agree about the number of senses of a given word, or how a word should be divided into senses. Therefore, it seems a bit too much to demand that the computer should be able to do it with a very high accuracy.

More problems come up with the definition of the semantic fields themselves, and assigning words to them. As was pointed out before, there is no consensus about what kinds of categories should be used. In addition, as Wilson & Thomas (1997:58-59) mention, many words seem to belong to more than one category. For example, *sportswear* belongs justifiably to the categories *sports* and *clothing*. As they also point out, it seems that semantic

categories in people's heads are "fuzzy sets" rather than clear-cut categories. In other words, it is natural for concepts to be associated with more than one sphere of activity. It has been suggested that semantic relationships could be represented in the form of a network, or by cross-referencing them, or in many other ways than one-to-one match between word and category, but there are technical and other problems which make many of such representations hard to carry out in practice.

Yet one more problem is caused by idioms. Garside & Rayson (1997:189) give an example of this. A semantic tagger's lexicon or idiom list could include an idiom like *HAVE in view*, meaning "intend". It would be able to distinguish the different forms of *have* appropriately. However, in real-life data, the expression may not occur on its own: in may become *has it in view* or *having several different aims in view*. For cases like this, parsing can prove to be the answer: it helps the computer to determine which words belong together and should possibly be viewed as a single unit.

This chapter has dealt with semantic annotation. The beginning of the chapter gave examples of uses for semantic annotation. The next section described how to add semantic annotation: what are the chances of doing it automatically, what kind of things a program could take into account and how to define semantic categories. The last section was about problems that concern semantic annotation.

Compared to part-of-speech-tagging and parsing, semantic annotation is not very common. Neither is the kind of annotation that will be discussed in the following chapter: anaphoric annotation.

3.8 Anaphoric annotation

Anaphoric annotation has to do with pronoun references in text (McEnery & Wilson 1996:52). The aim of anaphoric annotation is to make explicit what pronouns and other referring expressions are referring to. For example, in a sentence like *He took it there* there are three words (*he, it, there*) that refer to something that is not specified in this sentence. Instead, they refer to something that is named in other parts of the text or exists in the real world. Such relationships pass meaning "sideways" in a text (Garside, Fligelstone & Botley 1997:67), and anaphoric annotation tries to describe them.

The following extract is an example of anaphoric annotation.

Example 22: Anaphoric annotation

S.1 (0) The state Supreme Court has refused to release {1 [2 Rahway State Prison 2] inmate 1}} (1 James Scott 1) on bail .
S.2 (1 The fighter 1) is serving 30-40 years for a 1975 armed robbery conviction .
S.3 (1 Scott 1) had asked for freedom while <1 he waits for an appeal decision .
S.4 Meanwhile , [3 <1 his promoter 3] , {{3 Murad Muhammed 3} , said Wednesday <3 he netted only $15,250 for (4 [1 Scott 1] 's nationally televised light heavyweight fight against {5 ranking contender 5}} (5 Yaqui Lopez 5) last Saturday 4) .
S.5 (4 The fight , in which [1 Scott 1] won a unanimous decision over (5 Lopez 5) 4) , grossed $135,000 for [6 [3 Muhammed 3] 's firm 6], {{6 Triangle Productions of Newark 6} , <3 he said .

(i i) OR
[i...] enclose a constituent (normally a noun phrase) entering into an equivalence `chain'
<i indicates a pronoun with a preceding antecedent
>i indicates a pronoun with a following antecedent
{{i i} enclose a noun phrase entering into a copular relationship with a preceding noun phrase
{i i}} enclose a noun phrase entering into a copular relationship with a following noun phrase represents an anaphoric barrier, in effect, the beginning of a new text.

(http://www.comp.lancs.ac.uk/ucrel/annotation.html)

In this example, different referents are all given different numbers. For example, everything marked with number one (*James Scott, the fighter, he, his*) refers to the same person. The list above explains what the different kinds of brackets mean, although they are not of importance here: suffice it to say that

there are ways to indicate where an expression starts and ends, and ways to show whether an expression is referring further ahead in the text, or to something mentioned already.

Anaphoric annotation is very important in computational linguistics, for example in machine translation and in any attempts to make the computer understand natural language (Garside, Fligelstone & Botley 1997:67, McEnery & Wilson 1996:52). Pronouns carry so much meaning with them that it is impossible to understand natural text without understanding them. Fligelstone (1992:165) notes, interestingly enough, that anaphoric annotation may also have its uses in the field of speech synthesis. He writes that "The question arises whether prosody can be shown to have a predictable relationship with cohesive ties, and whether such a relationship can be used to improve algorithms for speech synthesis" (Fligelstone 1992:165). In other words, anaphoric annotation could improve the performance of speech synthesis systems.

Anaphoric annotation is, on the other hand, part of a larger field called **discourse annotation**. In addition to anaphoric relations, discourse annotation marks such features as information structure (center, theme, rheme), discourse roles of words and expressions, and discourse functions for sentences or groups of sentences (Garside, Fligelstone & Botley 1997:66). However, anaphoric relations are the easiest and best to start with. First of all, cohesive relationships in text are closely related to grammatical, syntactic and semantic annotation. Secondly, anaphoric annotation forms yet another step toward higher level annotations (Garside, Fligelstone & Botley 1997:66-67).

So far, very little discourse annotation has been done (Garside, Fligelstone & Botley 1997:66). It also has to be done mostly manually (McEnery & Wilson 1996:53). Garside, Fligelstone & Botley (1997:67-68) used a special program, XANADU, to help human annotators insert anaphoric tags. Their experience with anaphoric tagging was surprisingly positive: the task proved to be much easier than they had expected, the result more consistent and accurate. They also had a detailed annotator's manual, which was based on earlier decisions and helped to maintain consistency between the different annotators.

On an earlier publication, Fligelstone (1992) reports problems in developing the annotation scheme. Some types of word relationships had to be left out because not even a single annotator could tag them consistently. Such features included, for example, generic use of noun phrases (Fligelstone 1992:161). On the other hand, the annotation scheme was developed incrementally: the tagging conventions were introduced over a period of several months (Fligelstone 1992:155), so that different methods could be tested and there was no need to learn everything at once.

Since anaphoric references sometimes involve long stretches of text, the annotators had problems remembering everything that they should have kept in mind. This was taken into account when the program was designed, so that the computer could take care of remembering numbers and other details (Fligelstone 1992:156). Another point worth mentioning here is that they could not adapt their existing parsing and word-tag correction programs for this task: these did not allow the user to move around the text as freely as necessary for anaphoric annotation (Fligelstone 1992:155-156).

McEnery & Wilson (1996:53) point out that at this point, one of the aims of anaphoric annotation still is to produce data that could be used to train automatic taggers. It seems that the field of anaphoric annotation has not been tried extensively so far and there are yet no standard practices for it. In other words, it is still very much under development.

The following chapter will briefly describe another less common type of annotation: prosodic annotation. In addition, the annotation of spoken language corpora will be briefly discussed.

3.9 Prosody and spoken language annotation

This chapter will shortly deal with two types of annotation that both concern spoken language. The first one of them is **prosodic annotation**, which in fact has a much longer history than phonetic transcription (McEnery & Wilson 1996:54). The second is **spoken language annotation**, which often may be equal to phonetic transcription.

Note that these two do not fit into the "continuum" that has been presented here so far: whereas many of the other annotations tend to be cumulative, for example, semantic and anaphoric annotation may need POS tagging and parsing, prosodic and spoken language annotation have nothing to do with them. Spoken language transcription is, in fact, in the beginning of the continuum (spoken language needs to be transcribed before any further annotation can be applied to it), and prosody, too, is independent of other annotations.

3.9.1 Prosodic annotation

The first type of annotation to be considered in this chapter is called **prosodic annotation**. Prosodic annotation is the annotation of suprasegmental features of spoken language such as stress, intonation and pauses (McEnery & Wilson 1996:54). Leech, McEnery & Wynne (1997:88) list some uses for prosodically annotated corpora. These include the study of the grammatical composition of speech, the use of discourse markers, the occurrence of non-fluency phenomena, automatic speech segmentation and information flow in conversation. Such corpora can also be a help for speech recognition and synthesis. Leech, McEnery & Wynne (1997:89-90) also compare prosody in speech to punctuation in text: it is an essential part of it.

There are not many prosodically annotated corpora in the world. There are, however, two well-known corpora with prosodic annotation: the London-Lund corpus (LLC) and the Lancaster/IBM Spoken English corpus (SEC) (Leech, McEnery & Wynne 1997:86). The following is an example of prosodically annotated text from the London-Lund corpus.

Example 23: Prosodic annotation

```
1 8 14 1470 1 1 A 11 ^what a_bout a cigar\ette# .                    /
1 8 15 1480 1 1 A 20 *((4 sylls))*                                   /
1 8 14 1490 1 1 B 11 *I ^w\on't have one th/anks#* - - -             /
1 8 14 1500 1 1 A 11 ^aren't you .going to sit d/own# -              /
1 8 14 1510 1 1 B 11 ^[/\m]# -                                       /
1 8 14 1520 1 1 A 11 ^have my _coffee in p=eace# - - -               /
1 8 14 1530 1 1 B 11 ^quite a nice .room to !s\it in ((actually))#   /
1 8 14 1540 1 1 B 11 *^\isn't* it#                                   /
1 5 15 1550 1 1 A 11 *^y/\es#* - - -                                 /
```

The codes used in this example are:

```
# end of tone group
^ onset
/ rising nuclear tone
\ falling nuclear tone
/\ rise-fall nuclear tone
_ level nuclear tone
[] enclose partial words and phonetic symbols
. normal stress
! booster: higher pitch than preceding prominent syllable
= booster: continuance
(( )) unclear
* * simultaneous speech
-   pause of one stress unit
```

(McEnery & Wilson 1996:55)

McEnery & Wilson (1996:54-55) note that prosodic annotation does not have to cover all possible nuances of speech, but only the most prominent features. The rest can be inferred from the context, if that is necessary.

Indeed, there are practical reasons which prohibit extensive prosodic annotation. Prosodic annotation requires not only highly trained specialists to listen to speech recordings, but it is also very time-consuming and the compilation of such a corpus may take years (Leech, McEnery & Wynne 1997:89, McEnery & Wilson 1996:55). In addition, there is a problem with the "impressionistic nature of the judgements which are made" as McEnery & Wilson (1996:55) put it; i.e. different listeners may hear the same stretch of speech quite differently. For example, some people may hear only a fall in pitch, but others hear a rise in the end of the fall. Therefore, it is not easy to produce consistent prosodic annotation.

So far, prosodic annotation has been done manually and it has been considered to be impossible for computers (McEnery & Wilson 1996:55). However, there have also been attempts to insert prosodic annotation

automatically, which proved to be much faster than manual annotation (Leech, McEnery & Wynne 1997:90).

Finally, prosodically annotated corpora tend to have a problem with recoverability. It is hard to remove the tags, since many of them are in the middle of the words, and revert to the raw corpus. There are, however, attempts to apply TEI tags to prosodically annotated corpora, which could solve this problem. (McEnery & Wilson 1996:56, Leech, McEnery & Wynne 1997:90.)

3.9.2 Spoken language and speech corpora

Yet another types of corpora, which may be briefly mentioned here, are **speech and spoken language corpora**. The difference between these, according to Leech, McEnery & Wynne (1997:89) is that whereas a spoken language corpus consists of natural spoken language, a speech corpus has "laboratory" samples of speech. There may be, for example, the same word or sentence pronounced by speakers of different dialects. Speech corpora are useful for the study of speech synthesis, since it provides data of the details of pronunciation.

Due to character set problems, it is usually easier to use the normal alphabet than IPA codes for compiling spoken language corpora, although IPA codes have also been used. There is also the possibility to use TEI markup to represent unusual characters (McEnery & Wilson 1996:54). Even in the case of a normal character set, a system can be devised where normal characters correspond to certain IPA codes. Thus the corpus is comparable to an IPA-coded corpus, but it is easier to move from one computer to another, since no special character set is needed. (cf. McEnery & Wilson 1996:54)

Nevetheless, there are some problems with the annotations. First of all, as Leech (1997a:3) notes, the difference between representation and interpretation is not always clear. The transcribers first have to interpret what they hear in order to represent it in writing. There is also the problem that sounds rarely have clear boundaries, and what is phonetically the same sound may be quite different in different contexts (McEnery & Wilson 1996:54). In addition, it must be decided what is to be annotated and how – how to annotate overlapping speech, and whether to annotate such features as false starts,

hesitations, laughs, coughs and associated body language (McEnery & Wilson 1996:35-36).

Transcribed corpora are especially useful for people who want to study spoken language, but do not have access to the required equipment to study recorded speech in laboratory conditions (McEnery & Wilson 1996:54). However, since the compilation of such corpora needs expert human beings, it is time-consuming to prepare them.

This chapter has briefly looked at prosodic and spoken language corpora. Such corpora are not very common, and their compilation demands lots of resources and expertise. The following chapter will deal with even rarer types of annotation.

3.10 Other types of annotation

In addition to the types of annotation discussed so far, there are several other, less well-known types. They will be shortly discussed in this chapter.

Leech, McEnery & Wynne (1997:94-100) discuss **stylistic annotation**. In their study, they were interested in annotating speech and thought presentation and used tags for such features as direct speech, indirect speech, direct writing etc. The problems included, first of all, problems defining the tagset. Not only had there been none defined beforehand, so that they had to start from the beginning, but there were also problems at the theoretical level: stylistic categories are not as clearly defined as the categories for most other types of annotation (Leech, McEnery & Wynne 1997:95-96). In addition, there is no clear relationship between surface structure and the actual units that are to be tagged (Leech, McEnery & Wynne 1997:97-99), for example indirect speech cannot always be recognized from the way it is syntactically or orthographically presented. Different kinds of speech and thought representation may also overlap, or be embedded within each other (Leech, McEnery & Wynne 1997:97-98).

For these reasons, it is also hard to try to devise an automatic tagger for stylistic features. It might be possible to recognize automatically some categories, like direct speech, quite accurately. However, for example free indirect speech is typically characterized by the lack of clear indicators for it in text (Leech, McEnery & Wynne 1997:99-100). In any case, the recognition of stylistic features is to a great extent based on people's knowledge of literary texts and understanding of the world, and would require quite a lot of effort to be modelled by a computer (Leech, McEnery & Wynne 1997:100).

It is also possible to tag further **discoursal** or **pragmatic** features in a text. McEnery & Wilson (1996:52) list such units as "apologies", "hedges", "greetings", "politeness" and "responses" that could be tagged. Leech, McEnery & Wynne (1997:91) mention speech act types like "advisement", "confirmation", "question", "acknowledgement", "interpretation" and others that have been experimented with. As McEnery & Wilson (1996:52) point out, these kinds of tags have not been used very much not only because they have to be inserted manually, but also because there are problems in the linguistic,

theoretical part of the matter. The recognition of these categories is more or less a matter of interpretation, and different opinions abound.

Raumolin-Brunberg (1997) reports of gathering **sociolinguistic** information about a corpus. The corpus of Early English Correspondence consists of personal letters from 1420-1680. Information such as socio-economic status, education, sex and age of the author, date of writing and recipient were collected (Raumolin-Brunberg 1997:105-107). At the time of reporting, the information was not yet coded into the corpus, but kept in a separate database. However, the database allowed a search for, for example, information about people who lived at a certain place at a certain time and belonged to a specific occupation (Raumolin-Brunberg 1997:109-110).

Yet another type of annotation is **lemmatization**. McEnery & Wilson (1996:42) note that there are good and accurate software available for it, but it still has not been used very much. The problem here, too, is probably the surprising extent of ambiguity involved. For example, the words *move* and *moving* seem to belong to the same lemma, but in the connection *a very moving story* it can be questioned.

Leech (1997a:15) mentions that the **morphological structure** of words could be analysed. That would, no doubt, be of great use in the case of a language like Finnish, which has a complex morphology. Such analysis has been done, too.

McEnery & Wilson (1996:57) also mention **problem-oriented tagging**. It means that a researcher can add such annotation as is relevant to a particular research question. The tagset and other instructions for the application of tags can be chosen according to the needs of the study. For instance Meyer & Tenney (1993) developed an interactive tagging program that could be used to add problem-oriented tagging.

An example of this type of annotation could be, for example, what Leech (1997a:15) calls the annotation of learner corpora. He suggests that a researcher could tag classes of errors or features of non-native language behaviour to a corpus of learner language. This kind of corpus could prove to be a great help for studying second language acquisition.

This chapter has dealt briefly with some of the less well-known and relatively uncommon types of annotation. The next chapter, however, will move on to a rather different type of corpus annotation. The following chapter deals with alignment, which concerns bilingual and multilingual corpora.

3.11 Alignment

In this chapter I discuss alignment, which has to do with bilingual and multilingual corpora. Bilingual and multilingual corpora are often **translation corpora**. This means that the corpus contains both original texts and their translations into one or more languages. In order to fully take advantage of such a corpus, the texts need to be **aligned**.

The purpose of alignment is to show which parts of a text are translated by which parts of another text (Kay & Röscheisen 1993:121). It helps the user of a corpus to instantly retrive both the original sentence and its translation from the corpus. Alignment, from the computer's point of view, is some kind of markup in the texts which indicates which parts correspond to each other.

The type of alignment that seems to be the most common is called **sentence alignment**, whose purpose is to find corresponding sentences in the original text and its translation (Gale & Church 1993:75-76, Johansson et al. 1996:94). Another type of alignment is called **word alignment**, which is about finding word correspondences between the original text and the translation (Johansson & Hofland 1994:32, Gale & Church 1993:76). Many researchers seem to see word alignment as the final goal, and sentence alignment only a step towards it; however, sentence alignment itself can be very useful, and indeed it depends on the uses of the corpus whether word alignment needs to be considered at all.

This chapter mainly deals with sentence alignment. This is mostly because FECCS, the corpus used at the English department at the University of Jyväskylä, is only aligned by sentences, not by words. This chapter will, first of all, deal with the uses of aligned corpora and translation corpora in general; then, some basic principles of alignment are introduced. After that, three different types of methods of alignment are briefly presented, with the emphasis on the procedure used for FECCS. That is followed by discussion of the problems of alignment.

The alignment used for FECCS is based on the same principles as the alignment of the Norwegian ENPC corpus (English-Norwegian Parallel Corpus). It has been developed in Oslo and Bergen and later on adapted to

other language pairs in additions to English and Norwegian, including English-Swedish and English-Finnish.

3.11.1 Uses for translation corpora and aligned texts

Translation corpora can be used, for example, in the fields of lexicography, language teaching, translation, and in the study of language universals and language typology (Johansson & Hofland 1994:25). They are also important in the development of machine translation (Kay & Röscheisen 1993:121-122).

Translation corpora can be used, as the name implies, to study translations. The translations can be compared to the original texts, or maybe to other translations of the same text. As Johansson & Hofland (1994:25) say, such comparison brings out similarities and differences between languages, and shows what is general and what is language specific. Such corpora also make possible contrastive studies that examine and compare real texts instead of language systems of two languages.

Johansson and Hofland (1994:35-36) also give examples of more specific uses of a translation corpus. These include, for example, the study of translation equivalents and cases when the translation does not use the most obvious translation equivalent. One can also examine degree of correspondence between words and phrases, and cases where one sentence does not clearly correspond to only one sentence in the other text. Johansson and Ebeling (1994) note that it may also be of interest to examine original and translated texts in the same language, or compare translations in different languages and try to find similarities between them. Kay & Röscheisen (1993:121) mention that the bilingual corpora that exist nowadays also make possible statistical and other kinds of empirical studies of translation on a scale that was not possible before.

They also give an example of how corpora can be used in machine translation (Kay & Röscheisen 1993:121-122): as the computer is working on a translation, it can search the corpus to find sentences that are, in a way or another (e.g. syntactic structure), similar to the sentence that is being translated. The translations in the corpus have been made by human translators, and have thus been accepted by them as correct translations. The computer can

base its translation on corpus evidence. Much the same is mentioned by Gale & Church (1993:76); they claim that aligning sentences is just the first step, after which words could be aligned, for use in machine translation.

There exists criticism towards translation corpora. Johansson and Hofland (1994:25) note that some researchers tend to reject translation corpora, because the language of the translations may be different from the language of original texts in the same language. The claim is that such corpora do not "provide a good basis for contrastive studies". As Johansson & Ebeling (1994) point out, such texts "may reveal as much about the translators and the process of translation as about relationships between the languages involved". But of course, a corpus project dealing with translation corpora should take this into account. The uses of translation corpora may be limited in some ways, but its nature also makes some other types of research possible (e.g. the study of translationese, Johansson & Hofland (1994:26)).

Ebeling (1998:101) mentions that there are several browsers for bilingual corpora. The problem with them, however, tends to be that they were originally designed for a specific project and thus they may not be well suited for other kinds of texts and research questions. Some of these programs make automatic alignment themselves, whereas others require that alignment markup has been added beforehand. The following section explains how alignment works.

3.11.2 How alignment works

There are several things that can be taken into account when aligning sentences. Gale & Church (1993:76) note that the task of alignment is difficult, and may prevent people from taking advantage of bilingual corpora. They claim that the methods for alignment tend to be either unavailable, unreliable or computationally prohibitive. However, in recent years there have been also good experiences of alignment, including the FECCS project.

Johansson & Hofland (1994:30) list various methods that different projects have used to align texts. Generally, it seems that there are two factors that most alignment programs seem to take into account: sentence length and some kind of a bilingual dictionary, which includes key words in both

languages. Some alignment methods also use different kinds of statistical data to achieve good results.

The use of sentence length in alignment is based on the fact that longer sentences tend to be translated into long sentences, and short sentences into short sentences (Gale & Church 1993:75). Sentence length can be measured in the terms of both number of words and number of characters. However, both Gale & Church (1993:87-88, 90) and Johansson & Hofland (1994:30) note that measuring it by the number of characters is more reliable: somehow, the original sentence and its translation tend to be close to each other in terms of number of characters, but not necessarily in the number of words. Of course, it might be pointed out that both of these studies have dealt with the alignment of languages that are not too far apart from each other, such as English and Norwegian or English, German and French. When applied to languages that are not related to each other, the sentence length might not be such a good indicator of correspondence.

Many alignment programs also use a lexicon of some sort. ENPC uses an anchor word list of about 900 words. The words have been selected on the basis of their frequency and the degree of their equivalence in the two languages (Johansson & Hofland 1994:31). However, the alignment method used by Gale & Church (1993:89), for example, does not use any lexical information, and they also claim that such information is not necessarily needed. Their alignment worked well enough for their purposes without any kind of a word list.

One point worth mentioning here is that the success of alignment always depends on what kinds of texts are aligned. The following sections of this chapter will introduce three different methods of alignment. It must be noted that the texts used for them were rather different. Gale & Church (1993) were interested in the alignment of economic reports issued by the Union Bank of Switzerland, and the Canadian Hansards, which are parliamentary proceedings. Kay & Röscheisen (1993), in turn, experimented with two scientific articles that concerned physics. The Norwegian ENPC project, however, used prose fiction, which is rather different from the others. Especially the economic reports and parliamentary texts used by Gale &

Church are probably translated rather literally, so that the translator has not had much choice of words. The translations of prose fiction, however, sometimes include very free translations, and thus it cannot be aligned quite as easily as more formal texts.

For example, Kay & Röscheisen (1993:121) claim that the alignment of texts on the sentence and paragraph levels is easy and that there is usually no problems as to which sentences in translation correspond to sentences in the original text. However, as the experiences from the FECCS project have shown, this is not at all the case with fiction texts. Translators have the tendency to split sentences, combine them, leave them out entirely or move information to a different part of the text. It has also been noted that not even paragraphs tend to correspond to each other very well for a reason or another. This will be dealt with in more detail in section 3.11.6. The following sections, however, will introduce three different methods of aligning texts.

3.11.3 ENPC

The Norwegian ENPC project (English-Norwegian Parallel Corpus) has developed a program for adding sentence alignment to texts and their translations. The program takes into account many things, including sentence length, proper nouns, cognates, and punctuation marks (Johansson et al. 1996:95, Hofland 1995). The key to the workings of the program, however, are so-called anchor words, and the anchor list that they form (Hofland & Johansson 1998:87).

The anchor words are words that are either very common, or correspond well to each other in the two languages. The selection of words was partly based on intuition, and partly on manual matching of the texts (Johansson & Hofland 1994:31). The anchor word list includes about 900 words and their translations. The following is an example from the anchor word list of the FECCS project, which has used the same program as ENPC to align texts.

Example 24: Anchor word list

```
sing, sings, singing/ laul*
single / yksi*
sister* sisko*
sit* / istu*
situation* tilan*
sixteen* kuu*
sixth* kuud*
sixty*, kuu*
size* ko*
skin / iho*, nahk*
sleep*, slept / un*, nuk*
slow* / hi*
small, little / pien*
smaller pienem*
smallest pieni*
smel* haju*
smil* hymy*
smok* savu*
snow* lum*
```

The list always has the English word first, and then the possible Finnish translations following it. It may be noted here that some of the Finnish forms have been truncated rather a lot to give the program a chance to recognize inflected forms. Of course, this may also result into incorrect matches.

The program, basically, reads 15 sentences from each of the texts, and starts to look up the words in them from the anchor word list. Considering the anchor words and other things like punctuation marks and annotation that the sentences have in common, the program decides which sentences in the parallel texts must correspond to each other (Hofland & Johansson 1998:88-94). There usually is also an overlap of 5 sentences, so that the last five sentences of a group of 15 will be also the first five sentences of the next group of 15 to be processed (Hofland & Johansson 1998:88). This ensures that the program keeps on track even if the texts have a different number of sentences in them, as is often the case with translations.

The program makes, mostly, 1:1 matches so that one sentence in the original corresponds to one sentence in the translation. It can, however, make also 1:2 and 1:0 matches (Johansson et al. 1996:101). These are attempted if the difference in sentence length seems otherwise too high.

The success of the program has been rather good. As Hofland (1995) notes, it will keep on track even if 5-7 sentences are missing from the other text. The success rate (for English and Norwegian texts) is about 98 per cent

(Johansson & Ebeling 1994, Hofland & Johansson 1998:97). The program has also been tested on other language pairs, such as English-French, English-German, English-Polish, English-Finnish, English-Swedish and French-Norwegian, and the results are promising (Hofland & Johansson 1998:98).

The alignment of English and Finnish texts in the FECCS project has gone rather well. It may be questioned, however, whether the sentence length, for example, is a good indication of correspondence between an English and Finnish sentence. In majority of cases it probably is, but sometimes the program makes too many 1:2 or 1:0 alignments due to this, even though its eagerness to make such matches can be adjusted. For example, if the English sentence *Sir!* is translated into *Herra alikersantti!* the sentence length is rather different, and the program tries to find a 2:1 match, so that the Finnish sentence gets paired with two short English sentences. There may also be some problems with the word list, since Finnish lacks such common English words as articles and prepositions.

In order to go through the alignment program, the texts also need plenty of annotation to be added beforehand. These include tags that denote sentence boundaries, paragraph changes, chapter changes, and tags for chapter headings (Johansson et al. 1996:88-91). The ENPC project chose to use a tagging system that is based on TEI, because of the advantages to be gained by using an existing standard (Johansson & Ebeling 1994). The following is an example of tagged text from the FECCS corpus:

Example 25: Tagged text

```
<text>
<body>
<div1 type=part id=MA1.1>
<head>I Iron Lung</head>
<div2 type=chapter id=MA1.1.1>
<head>1</head>
<pb n=3><p><s>Time is not a line but a dimension, like the
dimensions of space.</s> <s>If you can bend space you can
bend time also, and if you knew enough and could move faster
than light you could travel backwards in time and exist in
two places at once.</s></p>
<p><s>It was my brother Stephen who told me that, when he
wore his ravelling maroon sweater to study in and spent
a lot of time standing on his head so that the blood would
run down into his brain and nourish it.</s> <s>I did n't
understand what he meant, but maybe he did n't explain it
```

```
very well.</s> <s>He was already moving away from the
imprecision of words.</s></p>

                    (Margaret Atwood: Cat's Eye (FECCS))
```

The <p> -tags denote paragraphs and the <s> -tags sentences. In the beginning of the above example, there are two -tags marking the beginnings of major divisions in the texts, and <head> -tags that mark chapter headings. Some of these tags can be added automatically, but some have to be inserted manually. All of them have to be checked manually at one point or another before alignment.

The alignment program needs these tags not only to recognize different parts of a text, but also to insert markup that will tell the corpus browser which sentences, in fact, do correspond. For this purpose, each sentence in the corpus gets a unique *id*-attribute and a *corresp*-value, which tells what sentence in the other text corresponds to it. The following is an example of aligned text from FECCS:

Example 26: Aligned text

```
<p id=MA1.1.1.p1>
<s id=MA1.1.1.s1 corresp=MA1T.1.1.s1>Time is not a line but
a dimension, like the dimensions of space.</s>
<s id=MA1.1.1.s2 corresp=MA1T.1.1.s2>If you can bend space
you can bend time also, and if you knew enough and could
move faster than light you could travel backwards in time
and exist in two places at once.</s></p>

                    (Margaret Atwood: Cat's Eye (FECCS))
```

In this example, it can be seen that the *id*-value for the first sentence is MA1.1.1.s1. This is a unique identity number that no other sentence in the database has. The *corresp*-value for this sentence tells that the Finnish translation for this sentence is called MA1T.1.1.s1. With the help of these values, it is easy to locate any given sentence in the corpus, and thus searching the corpus is fast.

However, as Ebeling (1998:104) points out, there is also a downside to this kind of annotation scheme. Since adding this kind of markup takes rather a long time, it is not suitable for texts that may be changed from time to

time. The texts need to be stored permanently, with no need to be manipulated or updated constantly.

In general, the experiences with the alignment program have been rather good. There are some problems with it, though. One of the problems is with short sentences, where the number of words is low. Johansson et al. (1996:102) give an example where the program has failed to pair the English sentence *Please!* with the Norwegian sentence *Vær så inderlig snill!* This is because the Norwegian sentence has many more words, including *inderlig* which is an intensifying adverb. Johansson et al. (1996:102) point out, however, that since short sentences are often questions or exclamations, punctuation marks may be of help to the alignment.

Another problem is formed by complex correspondences that can be found in very free translations (Johansson et al. 1996:102-103). For example, the end of one sentence in the original text might be actually the beginning of the following sentence in the translation (Johansson & Hofland 1994:35). Similarly, a translator may sometimes split sentences into several shorter ones, and since the alignment program is not able to do, for example, 1:3 or 2:3 matches, they will be aligned incorrectly.

The following example is from the FECCS corpus:

Example 27: Complex correspondences

```
<s id=JF1.1.s8 corresp='JF1T.1.s9 JF1T.1.s10'>A man in his
late twenties, in a dark bistre greatcoat, boots and a
tricorn hat, its upturned edges trimmed discreetly in silver
braid, leads the silent caravan.</s>
<s id=JF1.1.s9 corresp='JF1T.1.s11 JF1T.1.s12'>The
underparts of his bay, and of his clothes, like those of his
companions, are mud-splashed, as if earlier in the day they
have travelled in mirier places.</s>

<s id=JF1T.1.s9 corresp=JF1.1.s8> Äänetöntä kulkuetta
johtaa kolmeakymmentä lähentelevä mies, jolla on
tummanruskea, paksu päällystakki, saappaat ja
kolmikolkkahattu.</s>
<s id=JF1T.1.s10 corresp=JF1.1.s8>Hillitty hopeapunos
koristaa hatun ylös kaartuvia lieriä.</s>
<s id=JF1T.1.s11 corresp=JF1.1.s9>Hänen raudikkonsa vatsa ja
jalat, hänen omat vaatteensa samoin kuin hänen
kumppaniensakin ovat täynnä savitahroja.</s>
<s id=JF1T.1.s12 corresp=JF1.1.s9>Ilmeisesti he ovat
aikaisemmin päivällä kulkeneet kuraisessa maastossa.</s>

                    (John Fowles: A Maggot/Ilmestys (FECCS))
```

These examples have been manually corrected into two 1:2 matches, so that each English sentence corresponds to two Finnish sentences. The translator has not only split the original sentences, but also, in the case of the first English sentence, it has been split so that the contents of the second Finnish sentence corresponds actually to the middle of the first English sentence. This has been highlighted in boldface in the example. Also, the beginning of the first Finnish sentence, in fact, corresponds to the end of the first English sentence. Although this should not pose any problems to the alignment, since it does not consider the order of the words, it has been underlined in both the example sentences as a good example of free translation. In addition to the sentences being divided differently in the two texts, there may be problems regarding the vocabulary. The anchor word list certainly does not know words like *bistre* or *tricorn*, and *late twenties* has been translated into *kolmeakymmentä lähentelevä*. On the other hand, there are words that can probably be found in the wordlist, and the sentence lengths are surprisingly close to each other, if the lengths of the correct 1:2 alignments are compared.

The following example is also from the FECCS corpus:

Example 28: Free translation

```
<s id=SK1.1.s328 corresp=SK1T.1.s331>But now Tad was with
Cujo again, first hugging him extravagantly and then looking
closely at his face.</s>
<s id=SK1.1.s329 corresp=SK1T.1.s331>With Cujo sitting down
(his tail thumping on the gravel, his tongue lolling out
pinkly), Tad could almost look into the dog's eyes by
standing on tiptoe.</s>

<s id=SK1T.1.s331 corresp='SK1.1.s328 SK1.1.s329'>  Tad oli
kuitenkin ennättänyt jo koiran luokse ja katsoi sen
ruskeisiin silmiin varpaillaan seisten. </s>

                                    (Stephen King: Cujo (FECCS))
```

In this example, the translator has left out a rather surprising amount of information. It cannot be found in any of the surrounding sentences, either. On the other hand, the translator refers to the dog's brown eyes, which do not seem to be mentioned anywhere in the 13,000 word extract of the original text.

In this case, both the vocabulary and the huge difference in sentence lengths makes it more or less impossible for the alignment program to be able

to match the sentences correctly. Such cases have to be corrected manually. As Johansson & Hofland (1994:35) point out, the more the translator has deviated from the original, the more problems there will be. They also point out (1994:36) that fully automatic alignment will never be possible because of the complexity of the translation process.

This section has presented the method for aligning sentences that was originally developed for the Norwegian ENPC project, and which has been later on adapted to other languages. To a great extent, it relies on an anchor word list that includes key words in both of the languages that are being aligned. The following section, however, briefly describes a rather different approach to the task of alignment.

3.11.4 Gale & Church's method

Gale & Church (1993) developed a method for aligning sentences that is based on a statistical model of sentence lengths in characters. Their starting point was the fact that "longer sentences in one language tend to be translated into longer sentences in the other language, and that shorter sentences tend to be translated into shorter sentences" (Gale & Church 1993:75). They first tested their program on a trilingual corpus of economic reports issued by the Union Bank of Switzerland (English, French, German) and then tried it on a larger scale and aligned 90 million words of the so-called Canadian Hansards (Gale & Church 1993:75). The Hansards are parliamentary proceedings in English and French.

The program works in two steps. First it aligns the paragraphs, and then the sentences in them (Gale & Church 1993:79). It assigns a probabilistic score to each proposed pair of sentences, and then uses the scores to find the most likely correct alignment (Gale & Church 1993:78). It also makes 1:0, 1:2 and 2:2 matches between sentences in addition to the usual 1:1 matches (Gale & Church 1993:83).

Gale & Church (1993:78-79, 89) report that the program worked surprisingly well, considering that it is rather simple. The average error rate was 4.2%. If only the best-scoring 80% of the corpus was considered, the error rate dropped down to 0.7%.

The more complex the sentence correspondencies, the more errors there were with them. 1:1 matches were the easiest (only 2.6% error rate), 1:2 matches were somewhat harder (14%), and 2:2 matches had an error rate of over 30% (Gale & Church 1993:85). Curiously enough, all the 1:0 matches that the computer made were wrong. It missed all the real 1:0 cases, but found them where they did not exist (Gale & Church 1993:85).

Gale & Church (1993:89) claim that their system is fairly language independent. There are parametres in the program (concerning sentence length in the two languages) that can be adjusted (Gale & Church 1993:81-82). Even in their material, however, there was a difference between the success rate of English-German and English-French alignments: the English-French subcorpus had more errors in it (5.8%, compared to only 2.7% in the English-German corpus) (Gale & Church 1993:79, 85). The reason they give for this is that the original text was in German, so that in the English-French pair of texts both were translations to begin with. They also note that the overall low error rate is due to the high number of 1:1 sentence pairs in these texts, and that linguistically more different languages might be more problematic (Gale & Church 1993:85-86).

It can, indeed, be questioned how well their system would work with languages that are very different. In addition, what also must be pointed out is that the texts they used were rather rigid, formal reports. As Kay & Röscheisen (1993:140) note, the Hansards are known to be quite literal translations and they are not very hard to align. It does not seem likely that Gale & Church's method would work nearly as well with texts that have been translated more freely, like prose fiction.

McEnery & Oakes (1996), in fact, tested the performance of Gale & Church's method. They aligned texts of various genres (e.g. fiction texts, medical papers and newspapers) and tried several language pairs, including English-Polish, English-Spanish and Chinese-English. They noted that the results varied across genres and language pairs: with Chinese-English newspaper texts, the success rate went down to 54.5%. The explanation offered for this was the high number of 2:1 matches in the Chinese-English texts: although Gale & Church's method can deal with 1:1 alignments, the error rates

seem to be much higher for more complex matches (McEnery & Oakes 1996:213-214). McEnery & Oakes (1996), in fact, developed the program further by using information about cognates in it.

In all, their approach is rather different from the ENPC project. Whereas the ENPC alignment uses an anchor word list to find correct matches for sentences, Gale & Church's method does not use any lexical information at all. The following chapter, however, introduces one more different method of alignment.

3.11.5 Kay & Röscheisen's method

Kay & Röscheisen (1993) present a method of alignment that is similar to Gale & Church's method in the sense that it does not need any wordlists or other input aside from the texts themselves. Their alignment method is based only on internal evidence (Kay & Röscheisen 1993:122), in other words, it uses only information that can be found in the texts themselves.

Their method is rather interesting, since it uses partial word alignment in order to align sentences. It is based on the assumption that if a pair of sentences contains words that can be aligned, then the sentences can also be aligned (Kay & Röscheisen 1993:121-122). They note that full word alignment is difficult; however, there are some words, like technical terms and proper nouns, that can be paired quite easily (Kay & Röscheisen 1993:121). Their alignment method takes advantage of this.

Their program works basically by going through the texts several times, refining the alignments each time. It first selects sentences that it considers sure matches (e.g. the first and last few sentences in each text). Then it constructs a word list which includes likely word alignments based on the existing sentence alignments. On the basis of the word list, then, it can go on to make more sentence alignments, refine the word list again, and so on. Kay & Röscheisen note that it takes four or five passes through the texts for correct sentence alignment, and the program achieved something like 96% accuracy on their material. (Kay & Röscheisen 1993: 122-123, 138-139). They used scientific articles from *Scientific American* and their German translations (Kay & Röscheisen 1993:130).

Kay & Röscheisen (1993:140-141) claim that the texts they used were hard to align, and that their program produces good results even with relatively free translations. It seems, however, that they are comparing the ease of the task to the alignment of the Hansards, which, as noted before, are rather literal translations. It is hard to believe that translations of scientific texts could still be considered "free translations" if compared to translations of prose fiction. However, without seeing the texts, it is not possible to make definite claims; translations of scientific texts might be rather free, too, depending on what the translation is used for.

In the previous sections I have introduced three different ways to align parallel texts. They are all only examples of different methods that exist, and no doubt they suit certain kinds of texts better than others, and have problems that are specific to the relevant method of alignment. In the following section, however, I discuss some problems that are common to the task of alignment in general.

3.11.6 Problems with alignment

Alignment can be done on several different levels. Texts consist of parts, chapters, paragraphs, sentences, words, and so on. The ease of alignment more or less directly depends on the level that is being dealt with. It can be presumed that there are probably no huge problems with the level of chapters, for example. However, the lower the level of the units that are being dealt with, the more difficult the task of alignment will be.

Johansson & Hofland (1994:29) note that there is a good correspondence between higher-level units such as parts of books or chapters, and even paragraphs. Kay & Röscheisen (1993:121) also claim that aligning texts on paragraph level is easy. Gale & Church's method, indeed, relied on paragraph boundaries: the performance of their system degraded noticably when they tried to use it without paragraph boundaries as their starting point (Gale & Church 1993:88). However, as Johansson et al. (1996:91-92) note, there tend to be differences in paragraph divisions between texts and their translations. This kind of differences may be found, for example, in the representation of direct speech in prose fiction.

Kay & Röscheisen (1993:121) even go as far as to suggest that "there is rarely much doubt as to which sentences in a translation contain the material contributed by a given one in the original". To a human reader, after a careful examination of the texts, that may be true. However, the computer cannot make such assessments of freely translated fiction texts very easily. The translator may have combined or separated parts of sentences rather freely, or left out part of the information of the original text. The following is an example from the FECCS corpus. In it, the translator has made four sentences out of the original one sentence, even though it is otherwise rather literally translated:

Example 29: Differences in sentence division

```
<s>Tarkoitan, etten ole kauhistunut hänen tavastaan hallita,
pikemminkin olen hänen hallintonsa innokas kannattaja, mutta
silti. . . minä kerään ainoastaan tietoja, vertailen,
seulon, teen johtopäätöksiä.</s>

<s>I do n't mean I 'm disturbed by his performance.</s>
<s>I 'm actually rather an enthusiastic supporter of his
administration, but nevertheless ....</s>
<s>All I 'm doing is collecting information.</s>
<s>I form comparisons, I sift, I make inferences.</s>

        (Arto Paasilinna: Jäniksen Vuosi/The Year of the
        Hare (FECCS))
```

Some problems in sentence division also turn up because the translator has used another punctuation mark than the original writer, or otherwise somehow changed the typography of the text. The above example has an example of this, too, when the sentence following the three full stops starts with a capital letter in English, but not in Finnish. A stretch of text may look like several sentences in one text, but only one huge sentence in another because of these kinds of differences. This of course is not a problem for a human reader, but the computer always has some rather strict basis for deciding where one sentence ends and another one starts. Especially the semicolon tends to cause problems; where one text ends the sentence with full stop, the other has a semicolon there and the sentence continues onwards.

Some differences in sentence division are caused by grammatical differences in the languages. Johansson & Ebeling (1994) give the example of *ing*-clauses in English (e.g. *Jasper drank down his cup at once, and sat looking*

at the thermos, wanting more). Such clauses are often translated into independent sentences in Norwegian. The same kind of examples can be found in Finnish, too.

Plenty of differences in sentence division can also be found in connection of direct speech. The following is an example from the FECCS corpus.

Example 30: Differences in sentence division (2)

```
<s>"So," Lizzie had said.</s>
<s>"It means a careful Christmas." </s>

<s>"Eli jouluna täytyy olla törsäämättä", Lizzie oli
sanonut.</s>

        (Joanna Trollope: A Spanish Lover/Espanjalainen
        rakastaja (FECCS))
```

This is a typical example of a case where the translator has had to change the division of sentences in order to create a good translation that sounds natural in Finnish. These seem to be quite common with direct speech.

These types of problems with sentence divisions are more common with some text types than others. Johansson & Hofland (1994:29) note that for example legal texts tend to have sentence-by-sentence correspondence, whereas literary texts show much more differences. This can be seen in the FECCS texts, too: the few European Union directives that are part of the corpus have just a few sentences that have been divided or combined, whereas some fiction texts are more or less filled with 1:2 matches.

In this chapter I have explained what alignment is and introduced three different methods to align parallel texts. The alignment method used by the Norwegian ENPC project relied on a word list that is specific to the languages that are being aligned, whereas the two other methods did not use any external information at all, but only evidence that could be found in the texts themselves. Gale & Church's program compared sentence legths in the original text and the translation, and calculated probabilities to find out which sentences are likely to be pairs. Kay & Röscheisen, however, used possible word correspondences to align sentences correctly.

Each of these systems has its advantages: they are all suitable for certain types of texts and situations. The ENPC alignment requires lots of work before the sentences can be aligned, because markup needs to be added beforehand and a wordlist for the relevant languages needs to be constructed. The other two do not require such amount of preparations in advance. However, the ENPC system was also the most reliable of these. Although the success rates of the programs were not so different, it may be noted that the ENPC alignment was probably applied to rather more difficult texts that the other two. Gale & Church, indeed, say that their method could be recommended as a "first pass", and that for more accurate results some lexical information should be given to the program (Gale & Church 1993:87, 90). However, if there is a need to have only a quick alignment that does not need to be as accurate as possible, their method will work well enough. Note also that at least McEnery & Oakes (1996) have updated the program so that it uses a certain amount of lexical information, too.

There is, indeed, the question of what type of alignment is actually needed for a given project. The depth of alignment should always depend on the uses of the corpus: in some cases, simple paragraph alignment could be enough. There is necessarily no need to go for sentence alignment, let alone word alignment. Although this chapter does not explain word alignment, one may encounter references to it when reading about aligned corpora. It simply means, basically, that correspondences of individual words are made explicit. Word alignment is useful in machine translation and other such systems, where the computer needs to find word correspondencies.

The last section of this chapter considered the problems with alignment. These mostly have to do with the fact that, regardless of what some researchers claim, paragraphs and sentences rarely form 1:1 matches all the time. The section gave examples of cases when sentence divisions are not identical in the original and the translation. The reasons for this are many: some have to do with grammatical differences of the two languages, whereas in some cases it is simply a matter of the translator's style and preference.

3.12 Summary

In this part of the present paper I have discussed the field of corpus annotation. First of all, I considered general aspects of corpus annotation. This included general consideration of the uses of corpus annotation, examples of what annotation may look like, and general guidelines for annotation in the form of Leech's seven maxims of annotation.

Then, chapter 3.2 described how to get texts and how to computerize them. Chapters 3.3 and 3.4 were about standards, the need for them and what kinds of efforts there are towards standardization. Chapter 3.4 also introduced an existing standard, SGML, and the TEI guidelines that are specifically for encoding texts in the humanities.

Chapter 3.5 introduced part-of-speech annotation. Part-of-speech annotation, also known as POS tagging or grammatical tagging, aims to give each word in a corpus a word-class tag. The chapter explained the workings of an automatic tagger, considered different kinds of taggers and tagsets, and presented problems that are common in POS tagging. The problems include idiomatic sequences of words and other word combinations, ambiguous words and the fact that one hudred per cent accuracy does not seem to be possible even in POS tagging, even though it is the most common and easiest of annotations. POS tagging of languages other than English was also briefly considered.

Chapter 3.6 was about parsing, which is also known as syntactic annotation. In parsing, the sentences in a corpus are syntactically analysed and annotated so that they could be presented as syntactic trees. The chapter explained how an automatic parser works, and explained what kinds of theoretical models have been used with them. The end of the chapter discussed the problems with parsing, which mostly have to do with the definition of the rules that should be used, and their application to natural language.

Chapter 3.7 discussed semantic annotation, which is not nearly as common as the two previous ones. In practice, semantic tagging usually means that words are assigned to relevant semantic categories, which try to define word senses. The chapter considered both technical issues of applying semantic annotation, and the issue of defining the semantic categories. In the end, some

problems were mentioned, many of which have to do with the fact that in the real world, semantic categories are not so clearly defined and therefore are hard to model with a computer.

Chapter 3.8 briefly discussed anaphoric annotation. Anaphoric annotation tries to make pronoun references in a text explicit. Although it has not been used much yet, anaphoric annotation may prove to be significant, due to its great importance in computational linguistics and natural language processing.

Chapter 3.9 examined two kinds of annotation which are different from the previous ones: prosodic and spoken language annotation. Prosodic annotation is about adding information of such suprasegmental features of speech as stress, intonation and pauses. Spoken language annotation deals with the problems of representing speech in writing, and deciding what kinds of features should be transcribed. In case of both of these, there is the problem that they have to be done manually, and require experts who are used to listening to speech and can transcribe it as accurately as possible.

Chapter 3.10 considered other, even rarer types of annotation. These included stylistic annotation, discoursal and pragmatic annotation, sociolinguistic annotation, lemmatization, morphological annotation and problem-oriented tagging. All of these are quite rare and tend to lack tested conventions for their application.

Chapter 3.11 told about alignment. Alignment means that the original text and its translation in a bilingual or multilingual corpus are tagged so that corresponding sentences can be found easily. There are different types of algorithms for aligning texts. Most of them take advantage of factors such as sentence length, language-specific lexicons or probabilistic information. Alignment can also be done on many different levels: for some purposes, paragraph alignment may be enough, whereas other uses (such as natural language processing) need word alignment.

As has been noted many times before, it can be seen that the different types of annotations tend to build on each other and need each other as their basis. In the beginning of the continuum, there is part-of-speech tagging: using it is

fairly secure, since there are agreed conventions for doing it and the process is mostly automated. In the other end of the continuum, there are some of the less common annotations such as stylistic annotation: using them is less secure, since there are no agreed-upon methods and predefined categories of tags for applying them, they are not highly automated and the problems involved are not well known (Leech, McEnery & Wynne 1997:100-101). Of course, the conventions for applying certain kinds of tags can only develop through many experiments and the combined effects of several research projects. Therefore, one should not avoid the less common types of annotation, if there is the need and resources to go for them.

Many of the types of annotation presented here do indeed seem tempting. However, getting an annotated corpus is not fast, and it might cost quite a lot, too. As Hockey (1997:107) points out, the usefulness of a corpus increases as it gets more markup into it, but at the same time, it is more expensive to create. Even in the case of automatic tagging the time and money required to complete the corpus may be astounding, if the whole process is considered. Even though the computer might do its task rather fast, there are other parts such as scanning, proofreading, and post-checking the tags that can take a very long time.

The division of tasks between the computer and human beings is certainly something that has to be considered, too. The computer is faster, but it cannot handle well tasks that require interpretation or knowledge of the outside world. On the other hand, the errors the computer makes are at least consistent: human beings make mistakes out of boredom and tiredness, and make different interpretations of the same kind of examples (cf. Baker 1997:243-244). In addition, on one hand, the computer does not demand to be paid for the task; on the other hand, the programs it needs may be hard to get, expensive or may have to be programmed for the purpose, which of course costs money.

In addition to the question of speed and money there is the question of what is even possible. As has been noted before, it may be easy to define categories that would be useful to the user of the corpus, but then, might be more or less impossible to annotate within reasonable time and cost. If one is willing to annotate everything manually, one can try to define as fine-grained

categories as one wants to; however, practically, fully manual insertion of tags is usually not plausible, and compromises have to be made. The computer can only distinguish certain features of the text automatically, and one has to accept this at the moment. It is not certain that these problems will be easily solved in the future, either, since it does not seem that computers will be able to deal with such tasks that demand extensive knowledge of the outside world any time soon.

In addition to these complications, there is the fact that learning the annotation scheme takes quite a lot of time for the humans involved. Whether the annotation is done manually or automatically, the humans dealing with it need to understand what the tags mean and how they are supposed to be applied. Learning to understand the annotation in use may demand quite a lot of effort, especially for people who have not dealt with computers and programming very much before. Although annotating a corpus does not require programming skills, such skills are an advantage. The annotations are easier to read and the principle behind them easier to comprehend, if one is used to the way things are presented to the computer.

It is worth noting that depending on the type of annotation, there are commercial services available to do the tagging. A quick browse through the Internet seems to suggest that there are actually quite a few groups who do part-of-speech annotation or parsing. For example, the University of Lancaster (http://www.comp.lancs.ac.uk/ucrel/claws/) provide part-of-speech tagging services.

However, whether the annotation is performed automatically or manually, there has to be some kind of annotation manual, documentation which explains all the tags that are used. The manual has to not only name all the possible tags, but also explain how they have been used, and what kinds of decisions have been made in problematic cases. The manual does not have to be complete to begin with, but it can be developed as the annotation project proceeds. The annotation manual is useful not only to the annotators themselves, but to the end users who want to understand how the corpus has been tagged and why certain decisions have been made. It is good to remember

to try to document everything that is done during a project: it can prove to be invaluable help to someone else, later on.

As Leech (1997a:15) says, "annotation is an open-ended area of research, which is very much under development". This chapter has tried to give a general idea of what kinds of annotations exist, and what are gradually emerging.

CONCLUSION

In the present paper, I have dealt with the question of corpus compilation. The present paper, first of all, had a general introduction to corpus linguistics; after that, Part Two and Part Three dealt with computer issues and annotation, respectively.

The first part told about computer corpora in general: what they are, how to use them, and what they can be used for. In the first part, I gave the reader a general introduction to the field of corpus linguistics. First, different types of corpora were introduced and classified. After that, several important terms were explained and illustrated. The terminology covered terms relevant in both the use and compilation of corpora. I also introduced three programs that serve as good examples of corpus software. In the end of the chapter, several uses for corpora were discussed.

The second part of the present paper dealt with questions related to computer hardware and software. The chapters in Part Two gave general advice on many computer-related issues: How to select suitable hardware and operating system, how to deal with character set problems, how to store the data and where to acquire corpus software. The chapters also brought out many possible problem points in the use of computers, such as moving the data between different types of computers.

The third part of the present paper introduced corpus annotation. In the third part, I began by introducing what corpus annotation is and how it is usually applied. Secondly, different ways of acquiring suitable texts were discussed. Thirdly, Part Three also gave general information about standards, and introduced an existing text encoding standard that is relevant in corpus linguistics. In the rest of Part Three I discussed different types of annotations. They were introduced in a rough order of their frequency; part-of-speech annotation, which was introduced first, is also the most common type of annotation. From there, the present paper proceeded through parsing, semantic annotation, anaphoric annotation, prosody and spoken language annotation to other types of annotation, some of them quite rare. In the end of Part Three I described alignment, which concerns bilingual and multilingual corpora. Alignment means that corresponding parts of an original text and its translation

are found, so that they can be examined simultaneously. In practice, this often means that specific annotation is added to the corpus to tell the computer which parts of the texts are corresponding.

The rest of the present paper will consider a few more points that are important, but belong here, in the end, rather than under any more specific headings. All of these points have to do with corpus compilation in general.

First of all, the percentage describing the success of, say, automatic corpus annotation, does not tell the whole truth. For example, many automatic part-of-speech taggers are reported to achieve around 98% accuracy. Of course, this means that 98% of the tags were correctly applied; however, it also means that two out of one hundred tags are wrong. To the users, in practice, that may actually seem quite a number of errors.

For example, imagine reading a book that has been proofread hastily. Let's say there is a typo or a misprint every ten pages. Statistically this is not very much, if the percentage of correctly typed words are counted; to the reader, however, the text gives a very bad impression of being clumsily written or badly checked.

The same may be true with a corpus. Although researchers seem to agree that some amount of errors is (statistically) acceptable, in practice, the errors may seem to leap out of the paper. No amount of post-checking is likely to remove all the errors: on one hand, some of the errors may be caused by ambiguous units that cannot be unambiguously tagged, and on the other, human beings make errors too. This is something that has to be accepted with corpora. Although there undoubtedly are some well-known corpora in the world that are almost error-free, this is only because they have existed for a long time and many people have worked with them. They have had both the time and other resources to root out all the errors. In a small research project, however, errors will be inevitable. They will occur in all stages of corpus compilation – in scanning, proofreading, tagging, alignment and post-checking. The existence of errors in corpora is something that has to be accepted, and it is useful to keep in mind how large a portion of the corpus does *not* have errors in it, after all.

Secondly, document *everything* that you do. This does not only mean the annotation scheme that part three of the present paper mentioned several times, it means everything: what kinds of scanner settings were used to achieve the best results, what formats the texts were saved in, what kind of directory structure the files were stored in, the principles of naming the files, and so on. Write down all complications: for example, were there problems moving the files from a computer to another? What happened to special characters when the file was converted to another format, or moved to a different type of a computer? How did you solve the problem of characters that are not part of the standardized Low ASCII character set?

Such documentation has many uses. It is important if new people come to work in the same project: they can avoid making the same mistakes. It can also be very useful to other projects that are only starting. It is likely that many projects have the same type of problems, for example with regard to character sets, and it is of great importance to hear about others' experiences and solutions. Besides, writing down everything means also that you do not have to remember all the little details: you can come back to your notes and check how a problem was solved the last time it occurred.

Thirdly, a piece of advice that I would like to give about using computers is that it is worth to learn to use ordinary software, such as file managing and word processing tools, properly. It sometimes seems that many people waste their time with computers because quite ordinary procedures, such as copying files from one place to another, takes far too much time. Personally, I frequently meet people who, for example, copy files one at a time instead of selecting all of them with a few clicks of the mouse and copying them all at once. As well, often people do not use word processors' copying and pasting functions as often as they could be used, do not know the shortcut keys for them, and do not have any idea of what macros are. By learning to use simple functions like them, the overall time of processing texts can be reduced drastically. For instance, manual tagging often means repeating the same codes all the time, and it is a waste of time to type them all over again if they could be made into macros so that one code would be behind a simple key combination.

Also, learn to take advantage of the functions that a word processor offers. For example the proofread tool should be used; it does not weed out all the errors and mistypings, but the computer is arguably adept at noticing certain kinds of mistakes that occur. Modern word processors include quite a few tools the usefulness of which may not be realized without trying them out.

Generally, it is worth the trouble to learn to use the computer and its software properly. Not everything has to be learned at once, of course; it is easiest to learn new procedures gradually. However, users should not think that they already know everything that they need to know. In many cases, people do not use programs as efficiently as they could because they simply do not know that something could be done faster. In the long run, it is worth the trouble to spend some time learning to use new tools; with time, it makes using computers much faster, easier and more fun. Of course, the overall efficiency of work is also improved, since more time is available for the real work as the commonplace tasks can be handled faster.

In the present paper, I have tried to give an overall picture of corpus compilation and introduce some general principles, alongside with examples that are likely to be relevant even later. Although the scope of the present paper is rather wide, it has not given very practical advice for many issues. The reason for this is that since computers and everything related to them develops rather fast, it would not be useful to start giving details of facts that will not apply a year later.

In practice, if you want to learn to do something that is presented here, get a hold of some more detailed information of the topic. For example, if you want to learn more about POS tagging, find an annotation manual that was written by a previous project. Even better, get hold of the people who did it and ask them what you need to know. Ask them if they have a tutorial that they used to train their annotators. Search for free demos of annotated corpora to see how they work: there are some corpora that can be accessed through the Internet, and hopefully, there will be more of them in time. Get to know as many corpora with POS tagging as possible, see what kind of annotation they use, and how the programs with them work. Even if you do not get a chance to

try them out yourself, there may be written reports of other people's experiences. Then you can decide what kind of tagging would be the best for your corpus, and how to best achieve it.

In the present paper I have tried to collect under one heading all those bits and pieces about compiling a corpus that are relevant for any corpus project. Even though some of the information is rather theoretical, it should give a basis from which one can go on to find further information of specific types of corpora, and of more practical matters. I hope that the present paper proves to be useful for anyone who has an interest towards corpora and their compilation.

REFERENCES

Aijmer, K., B. Altenberg and M. Johansson (eds.) 1996. *Languages in contrast. Papers from a Symposium on Text-based Cross-linguistic Studies, Lund 4-5 March 1994.* Lund: Lund University Press

Antworth, Evan L. and J. Randolph Valentin 1998. Software for doing field linguistics, in Lawler and Dry (eds.) 1998, 170-196.

Armstrong, Susan (ed.) 1993. *Using Large Corpora.* London: A Bradford Book, The MIT Press.

Bailey, Richard W. and Anne Curzan 1997. The Diary of Henry Machin: An electronic text, in Hickey et al. (eds.) 1997, 25-32.

Baker, John Paul 1997. Consistency and Accuracy in Correcting Automatically Tagged Data, in Garside et al. (eds.) 1997, 243-250.

The Bank of English – Questions and Answers. <http://titania.cobuild.collins.co.uk/boe_info.html> undated (Accessed 25 Nov. 1997).

Barnbrook, Geoff 1996. *Language and Computers – A Practical Introduction to the Computer Analysis of Language.* Edinburgh: Edinburgh University Press.

Bateman, Jeremy, Jean Forrest and Tim Willis 1997. The Use of Syntactic Annotation Tools: Partial and Full Parsing, in Garside et al. (eds.) 1997, 166-178.

Biber, Douglas, Edward Finegan and Dwight Atkinson 1993. ARCHER and its challenges: Compiling and exploring a representative corpus of historical English registers, in Fries et al. (eds.) 1993, 1-14.

Biber, Douglas, Susan Conrad and Randi Reppen 1994. Corpus-based Approaches to Issues in Applied Linguistics, *Applied Linguistics* 15/2, 169-189.

Biber, Douglas, Susan Conrad and Randi Reppen 1998. *Corpus Linguistics: Investigating Language Structure and Use*. Cambridge: Cambridge University Press.

Burnard, Lou 1992. Tools and Techniques for Computer-assisted Text Processing, in Butler (ed.) 1992, 1-28.

Butler, Christopher S. (ed.) 1992. *Computers and Written Texts*. Oxford: Blackwell.

Byers, Robert A. 1985. *Introduction to UNIX System V*. Culver City, California: Ashton-Tate.

CETH (Center for Electronic Texts in the Humanities) (homepage). <http://www. ceth.rutgers.edu> undated (Accessed 8 Jun. 1999).

CETH newsletter, spring 1995. <http://www.ceth.rutgers.edu/Ceth/newsletter/news31/ocr.html> undated (Accessed 9 Dec. 1998).

Collins COBUILD (homepage). <http://titania.cobuild.collins.co.uk> undated (Accessed spring 1999).

Collins COBUILD. *Information about the CobuildDirect Service.* <http://www.cobuild.collins.co.uk/direct_info.html> undated (Accessed 8 Jun. 1999).

Conexor Oy (homepage). <http://www.conexor.fi> 1997-1998 (Accessed 8 Jun. 1999).

Deitel, H. M and P. J. Deitel 1994. *C – How to Program (Second edition)*. New Jersey: Prentice-Hall.

Dougherty, Dale and Tim O'Reilly 1988. *DOS meets UNIX*. USA: O'Reilly & Associates, Inc.

Dry, Helen Aristar and Anthony Rodrigues Aristar 1998. The Internet: an introduction, in Lawler and Dry (eds.) 1998, 26-61.

Ebeling, Jarle 1998. The Translation Corpus Explorer: A browser for parallel texts, in Johansson and Oksefjell (eds.) 1998, 101-112.

Eloranta, Jussi, Seppo Kallio and Matti Levänen 1994. *Unix – käyttäjän opas (3. uudistettu painos). Käyttäjän opas N:o 16.* Jyväskylä: Jyväskylän yliopiston ATK-keskus.

European Union 1995. *The Green Paper on Copyright and Related Rights in the Information Society.* <http://www.ispo.cec.be/infosoc/legreg/com95382.doc> 1995 (Accessed spring 1999).

The FECCS (Finnish-English Contrastive Corpus Studies) corpus, compiled at the Department of English at the University of Jyväskylä.

Fligelstone, Steve 1992. Developing a scheme for annotating text to show anaphoric relations, in Leitner (ed.) 1992, 153-170.

Fligelstone, Steve, Mike Pacey and Paul Rayson 1997. How to Generalize the Task of Annotation, in Garside et al. (eds.) 1997, 122-136.

Fries, Udo, Gunnel Tottie and Peter Schneider (eds.) 1994. *Creating and Using English Language Corpora: Proceedings from the Fourteenth International Conference on English Language Research on Computerized Corpora, Zürich 1993.* Amsterdam: Rodopi.

Gale, William A. and Kenneth W. Church 1993. A Program for Aligning Sentences in Bilingual Corpora, in Armstrong (ed.) 1993, 75-102.

Garside, Roger and Nicholas Smith 1997. A Hybrid Grammatical Tagger: CLAWS 4, in Garside et al. (eds.) 1997, 102-121.

Garside, Roger and Paul Rayson 1997. Higher-level Annotation Tools, in Garside et al. (eds.) 1997, 179-193.

Garside, Roger, Geoffrey Leech and Anthony McEnery (eds.) 1997. *Corpus Annotation: Linguistic Information from Computer Text Corpora.* London: Longman.

Garside, Roger, Steve Fligelstone and Simon Botley 1997. Discourse Annotation: Anaphoric Relations in Corpora, in Garside et al. (eds.) 1997, 66-84.

Glossary of corpus linguistics. <http://www-clg.bham.ac.uk/glossary.htm> undated (Accessed 11 Mar. 197).

Granger, Sylviane 1996. From CA to CIA and back: An integrated approach to computerized bilingual and learner corpora, in Aijmer et al. (eds.) 1996, 37-51.

Greenbaum, Sidney 1993. The Tagset for the International Corpus of English, in Souter & Atwell (eds.) 1993, 11-24.

Guthrie, Louise 1993. A Note on Lexical Disambiguation, in Souter & Atwell (eds.) 1993, 227-238.

Healey, Antonette diPaolo 1997. Wood-gatherers and cottage builders: Old words and new ways at the Dictionary of Old English, in Hickey et al. (eds.) 1997, 33-46.

Hickey, Raymond, Merha Kytö, Ian Lancashire and Matti Rissanen (eds.) 1997. *Tracing the Trail of Time: Proceedings from the Second Diachronic Corpora Workshop*. Amsterdam: Rodopi.

Hirvonen, Pertti 1994. *Ohjelmoinnin alkeet C-kieltä käyttäen. Oheismateriaalia itseoppijalle. Opetusmonisteita OM-07*. Jyväskylä: Tietojenkäsittelytieteiden laitos.

Hirvonen, Pertti 1995. *Ohjelmoinnin peruskurssi C-kieltä käyttäen. Opetusmonisteita OM-08*. Jyväskylä: Tietojenkäsittelytieteiden laitos.

Hockey, Susan 1998. Textual Databases, in Lawler and Dry (eds.) 1998, 101-137.

Hofland, Knut 1995. *A Program for Aligning English and Norwegian Sentences*. Paper from ALLC/ACH'95, Santa Barbara, July 11-15, 1995.

Hofland, Knut 1997. *From sentence alignment to word alignment.* Paper presented at the Fourth Nordic Symposium on text-based contrastive studies, 25-27 April 1997.

ICAME (International Computer Archive of Modern and Medieval English) (homepage). <http://nora.hd.uib.no/icame.html> undated (Accessed 8 Jun. 1999).

Iomega (homepage). <http://www.iomega.com> 1999 (Accessed spring 1999).

Janssen, Sylvia 1992. Tracing cohesive relations in corpora samples using dictionary data, in Leitner (ed.) 1992, 143-152.

Johansson, Stig 1986. *The Tagged LOB Corpus: User's Manual.* Bergen: Norwegian Computing Centre for the Humanities.

Johansson, Stig and Jarle Ebeling 1994. *The English-Norwegian Parallel Corpus: Introduction and Applications.* Paper submitted to The XXVIII International Conference on Cross-Language Studies and Contrastive Linguistics, Rydzyna, Poland, December 15-17, 1994.

Johansson, Stig and Knut Hofland 1998. The Translation Corpus Aligner: A program for automatic alignment of parallel texts, in Johansson and Oksefjell (eds.) 1998, 87-100.

Johansson, Stig and Signe Oksefjell (eds.) 1998. *Corpora and Cross-linguistic Research: Theory, Method and Case Studies.* Amsterdam: Rodopi.

Johansson, Stig, Jarle Ebeling and Knut Hofland 1996. Coding and aligning the English-Norwegian Parallel Corpus, in Aijmer et al. (eds.) 1996, 87-112.

Jyväskylän yliopiston kirjaston tutkielmapankki (homepage). <http://docuweb.jyu.fi> undated (Accessed 8 Jun. 1999).

Kahrel, Peter, Ruthanna Barnett and Geoffrey Leech 1997. Towards Cross-Linguistic Standards or Guidelines for the Annotation of Corpora, in Garside et al. (eds.) 1997, 231-242.

Kay, Martin and Martin Röscheisen 1993. Text-Translation Alignment, in Armstrong (ed.) 1993, 121-142.

Kiianmies, Matti 1995. *Windows 3.11 Askel askeleelta.* Jyväskylä: Suomen ATK-kustannus Oy.

Kivimäki, Jyrki and Kimmo Rousku 1996. *Windows 95 – tehokäyttäjän opas, osa I.* Jyväskylä: Suomen ATK-kustannus Oy.

Kurzweil, Ray 1996. *Why I am Building Reading Machines Again.* <http://www.kurzweiledu.com/readingmachines1.html> 19 Jun. 1998 (Accessed 9 Dec.1998).

Lauridsen, Karen M. 1996. Text corpora and contrastive linguistics: Which type of corpus for which type of analysis? in Aijmer et al. (eds.) 1996, 63-71.

Lawler, John and Helen Aristar Dry (eds.) 1998. *Using Computers in Linguistics – A Practical Guide.* London: Routledge.

Lawler, John M. 1998. The Unix™ language family, in Lawler and Dry (eds.) 1998, 138-169.

Leech, Geoffrey 1993. Corpus Annotation Schemes, *Literary & Linguistic Computing 8(4)*, 275-281.

Leech, Geoffrey 1997a. Introducing Corpus Annotation, in Garside et al. (eds.) 1997, 1-18.

Leech, Geoffrey 1997b. Grammatical Tagging, in Garside et al. (eds.) 1997, 19-33.

Leech, Geoffrey and Elizabeth Eyes 1997. Syntactic Annotation: Treebanks, in Garside et al. (eds.) 1997, 34-52.

Leech, Geoffrey and Steven Fligelstone 1992. Computers and Corpus Analysis, in Butler (ed.) 1992, 115-140.

Leech, Geoffrey, Tony McEnery and Martin Wynne 1997. Further Levels of Annotation, in Garside et al. (eds.) 1997, 85-101.

Leech's Maxim's of Annotation. <http://www.ling.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2maxims.htm> undated (Accessed 19 Aug. 1998).

Leitner, Gerhard (ed.) 1992. *New Directions in English Language Corpora: Methodology, Results, Software Developments*. Berlin: Mouton de Gruyter.

Manual to the Diachronic Part of the Helsinki Corpus of English Texts – Coding Conventions and List of Source Texts. <http://www.helsinki.fi/hum/eng/projects/manuaali.html> undated (Accessed 25 Nov. 1997)

Marcus, Mitchell P., Beatrice Santorini and Mary Ann Marcinkiewicz 1993. Building a Large Annotated Corpus of English: The Penn Treebank, in Armstrong (ed.) 1993, 273-290.

Mauranen, Anna 1997. Introduction to Corpus Linguistics. A course given at the University of Jyväskylä, Department of English, 3rd and 10th October 1997.

McEnery, Tony and Andrew Wilson 1996. *Corpus Linguistics*. Edinburgh: Edinburgh University Press.

McEnery, Tony and Michael Oakes 1996. Sentence and word alignment in the CRATER Project, in Thomas & Short (eds.) 1996, 211-231.

McEnery, Tony and Paul Rayson 1997. A Corpus/Annotation Toolbox, in Garside et al. (eds.) 1997, 194-208.

Meyer, Charles F. and Richard L. Tenney 1993. Tagger: An Interactive Tagging Program, in Souter & Atwell (eds.) 1993, 25-36.

Part-of-speech Annotation: An Example. <http://www.ling.lancs.ac.uk/monkey/ihe/linguistics/corpus2/2posex.htm> undated (Accessed 1 Dec.1998).

Patten, Terry 1992. Computers and Natural Language Parsing, in Butler (ed.) 1992, 29-51.

Pennington, Martha C. and Vance Stevens (eds.) 1991. *Computers in Applied Linguistics: An International Perspective*. Clevedon: Multilingual Matters Ltd.

Peters, Carol and Eugenio Picchi. *Bilingual reference corpora for translators and translation studies*. <http://www.ilc.pi.cnr.it/dbt/art_comp.htm> undated (Accessed 17 Mar. 1997).

Raumolin-Brunberg, Helena 1997. Incorporating sociolinguistic information into a diachronic corpus of English, in Hickey et al. (eds.) 1997, 105-118.

Rogers, Henry 1998. Education, in Lawler and Dry (eds.) 1998, 62-100.

Roponen, Seppo and Timo Harmo 1991. *Sinä ja tietokone: tietokonetietoa humanisteille*. Helsinki: Gaudeamus.

Sampson, Geoffrey 1991. Analysed Corpora of English: A Consumer Guide, in Pennington & Stevens (eds.) 1991, 181-193.

Sánches León, Fernando and Amalio F. Nieto Serrano 1997. Retargeting a Tagger, in Garside et al. (eds.) 1997, 151-165.

Simons, Gary F. 1998. The Nature of Linguistic data and the requirements of a computing environment for linguistic research, in Lawler and Dry (eds.) 1998, 10-25.

Sinclair, John 1991. *Corpus, Concordance, Collocation*. Oxford: Oxford University Press.

Smith, Nicholas 1997. Improving a Tagger, in Garside et al. (eds.) 1997, 137-150.

Souter, Clive and Eric Atwell (eds.) 1993. *Corpus-based Computational Linguistics*. Amsterdam: Rodopi.

Sperberg-McQueen C. M. and Lou Burnard 1994. *Guidelines for Electronic Text Encoding and Interchange (TEI P3) Vol. I.* Chicago-Oxford: ACH, ACL and ALLC.

Stubbs, Michael 1994. Grammar, Text and Ideology: Computer-assisted Methods in the Linguistics of Representation. *Applied Linguistics* 15/2: 201-223.

Thomas, Jenny and Mick Short (eds.) 1996. *Using Corpora for Language Research.* London: Longman.

UCREL. *CLAWS part-of-speech tagger.* <http://www.comp.lancs.ac.uk/ucrel/claws/> undated (Accessed 8 Jun. 1999).

UCREL. *Corpus Annotation.* <http://www.comp.lancs.ac.uk/ucrel/annotation.html> undated (Accessed 19 Aug. 1998).

W3 Corpora (homepage). <http://clwww.essex.ac.uk/w3c/> 1998 (Accessed spring 1999).

W3 Corpora. *Corpus Linguistics Software.* <http://clwww.essex.ac.uk/w3c/corpus_ling/content/software.html> June 1998 (Accessed 8 Jun. 1999).

Wilson, Andrew and Jenny Thomas 1997. Semantic Annotation, in Garside et al. (eds.) 1997, 53-65.

Wilson, Andrew and Paul Rayson 1993. The Automatic Content Analysis of Spoken Discourse: A Report on Work in Progress, in Souter & Atwell (eds.) 1993, 215-226.

WordSmith Tools 3.0 (English Language Teaching Catalogue, Oxford University Press). <http://www1.oup.co.uk/elt/catalogu/multimed/4589846/4589846.html> undated (Accessed 8 Jun. 1999).

WordSmith Tools 3.0 (screenshot: Concord: A Concordance). <http://www1.oup.co.uk/elt/catalogu/multimed/4589846/screen1/text1.html> undated (Accessed 8 Jun. 1999).

Voutilainen, Atro and Juha Heikkilä 1994. An English Constraint Grammar (ENGCG): a surface-syntactic parser of English, in Fries et al. (eds.) 1994, 189-199.

APPENDIX 1: CONCORDANCE OF *BLUE*

Concordance of the word *blue*, sorted by the first word to the right of the central word.

```
WordSmith Tools Concordancer          BLUE
16:43, 31.05.1999
134 entries in total. Windows ANSI format.
search word: BLUE
sort: 1st word to right then

    1 e, too. Women should never wear blue &dash; it 's too innocent a
    2  his eyes, that are of a vacant blue almost as if he were blind,
    3 an. I was a bit blue, and being blue always makes me satirical, s
    4 ole painted terracotta and deep-blue and Chinese-yellow, with pol
    5 indows smouldering. It was that blue and red, that blue especiall
    6 een, the newly cxtended kitchen blue and russet and cream. "Why i
    7 roadside diners from the 1950s. Blue and green neon spells the na
    8 before realising that the red, blue and orange computer men repr
    9 rowing all over them, green and blue and yellow. Kevin dared Jame
   10 droom where a truly magnificent blue and yellow parrot sat in a t
   11 e, painted grey. The carpet was blue, and the walls were baby blu
   12 xited with Gillian. I was a bit blue, and being blue always makes
   13  up the caterpillars, which are blue-striped, and velvety and coo
   14 en she pulled on a light fluffy blue angora jumper. She had thin
   15 see the country, and we sure in blue blazes are seeing it now. No
   16 f-naked body. Now she takes the blue bottle and moistens a corner
   17 ly away and reaches for another blue bottle. That has a goose qui
   18 mirror and one of the minuscule blue bottles, stoppered with a co
   19 ped pine of the dresser and the blue bowl of orange marigolds and
   20  else, and I said it out of the blue, but in my mind it was as if
   21 d wear the distinctive electric-blue cap bands, shoulder boards a
   22  woman, was resplendent in pale-blue chiffon and cloth of silver,
   23 e nephew fills his glass from a blue-and-white china decanter of
   24 mdash; she was wearing a bright blue dress from the early days of
   25 atalie. She had put on her best blue dress with the wide white co
   26  It was that blue and red, that blue especially, I felt in need o
   27 ky and flaxen-haired, with cold blue eyes and a sharp chin. I had
   28 ely pale oval madonna face with blue eyes and her hair was light-
   29  continue straining your lovely blue eyes and end up bumping into
   30 trudged through icy snow. Blank blue eyes stared at the old man w
   31 . And this old woman's freakish blue eyes had turned it into some
   32 t-chested and had pale skin and blue eyes and long fingers. Her h
   33 e? The gentle pity in the faded blue eyes robbed Mattie of the an
   34 ell, but at least the prominent blue eyes had not been veiled. Th
   35 s to them, even when his vacant blue eyes were on Farthing and he
   36 owever, he stared down with his blue eyes at the old table just b
   37 oyish looks, fluffy blond hair, blue eyes and shy smile, Billy wa
   38 he colour of her somewhat blank blue eyes &dash; I tell you, litt
   39 in the metropolis. But his baby blue eyes missed very little. Bef
   40 me through them, which made her blue eyes look huge. "You could n
   41 s her head; at which his vacant blue eyes look away from her brow
   42  accident," said Alice, and her blue eyes opened wide with fancie
   43 ng look in both pairs of bright blue eyes, she concealed them. Te
   44 sh face, the soft skin, the shy blue eyes, the warm damp lips. Al
   45 me man with wavy brown hair and blue eyes, straight nose and stro
   46 y was n't Daddy. He had Daddy's blue eyes, Daddy's dark brown hai
```

o dangerous. Cold and murderous blue eyes. He had really wanted t
straight nose and bright, pale blue eyes. She had never known an
edulously into a pair of watery blue eyes. "What you gapin' at?
rom remarkable, slanted, violet-blue eyes. Standing over her, he
t meeting. She had been wearing blue faded jeans, and a white swe
d clothes. Spitting that Maid's blue flame right in the face and
avelling through the air like a blue flame, killing her victims s
to boil on the hissing ring of blue-and-orange flames. "Twins,"
ug beside her. He was wearing a blue fleecy sleeping suit embroid
n — " She gestured at the blue frills of her nylon nightgow
nuine Austrian princess, a real blue-blood from the House of Habs
rward, widening her pale bright blue give-away impenetrable eyes
There are some small and corked blue glass bottles also; a comb,
ines of the olive-dun and grape-blue hills punctuated by saffron
d it? Flying, flying across the blue hills and crystal lakes and
joyous bells far off beyond the blue hills, beyond the green mead
r if it comes to that, but in a blue Honda he finds the keys tuck
ght eye was green, her left eye blue. I 'd seen a white cat like
had to unfreeze the pipes with blue-flames. In the beginning we
-green, lemon-yellow, and berry-blue. In its polished metal and P
t day in early July, the sky so blue it made his eyes ache. He re
better. Then, quite out of the blue, it is discovered by one of
simmering greens. Lovella wore blue jeans and a plain white tee
s, almost into a bonnet, by the blue kissing-ribbons beneath her
propped up in bed and wearing a blue lacy nightgown, smiled at hi
lid into her unconscious like a blue laser and exposed secrets th
eet encased in good, dull, dark-blue leather loafers. She knew ex
ght of space and blackness, the Blue Life Lounge huddles like a r
hat across the highway from the Blue Life Lounge is a motel where
8 In the Blue Life Lounge, a woman brushes
e highway from the motel to-the Blue Life Lounge and gets in his
e and my windows flickered with blue light, tree branches illumin
s and white stocks for morning, blue liveries with brass buttons
ist in his hand murmured to the blue-eyed man and the smiling, be
nt of Daddy's thumb, with faint blue markings on his carapace. Sh
, maybe nine by nine, with four blue molded-plastic chairs hooked
small piece of bread. "Blimmin' blue," muttered Tom to himself as
vanished round a corner where a blue neon sign said, "Bar El Nido
ves, just like that, out of the blue. No, usually if you want thi
ernoon sun against the crinkled blue of the sea and, breasting a
ing undefined; except the eyes. Blue, of course. Not very large a
. They were already known to be blue-whites, once also called Top
hristmas tree lights, yellow or blue or green. These are called "
hair dyed straw-blonde or baby-blue, or, even more startling aga
r feet were painted with indigo-blue patterns. She gave each of t
He wore dark corduroys, a navy blue pea jacket, and a fisherman'
wearing grey slacks and a dark-blue plaid shirt, packing our foo
sessed. Tonight he wore a faded blue polka-dot cravat at his thro
an sprint to a large silver and blue Road King, which he enters t
dress of dark blue satin, and a blue rose at the apex of her deco
herself between a lady in royal-blue ruffles with a fan and casta
if she could come get the navy blue rug from the dining room. "N
rug from the dining room. "Navy blue rug," Macon repeated. (He wa
wn her chin, it was matting her blue sailor dress, blood, oh dear
ess and wearing a dress of dark blue satin, and a blue rose at th
at was the Countess of Powis in blue satin, diamond-embroidered;
dinner at Dunbarton, out of the blue, she decided it could n't be
ter was easing off the bone and blue skin. "I am not from the Peo
on. A soft June morning with a blue sky and a gentle breeze. Six
bright for safety, with a pale blue sky arching much too high ab
the end of its tiny tunnel. Ice-blue sky, yellow dumps, black vel

ng clouds which had crowded out blue sky, leaving only an occasio
in white clouds sail across the blue sky. Small birds flew by ove
were differently shaped, large blue spheres with, above them, st
were to the wisdom of a yellow, blue-eyed spirit who had foreseen
d navy school uniform skirt and blue striped blouse, which passed
mdash; a small man waited, in a blue suit. He carried a placard w
ir driver, similarly dressed in blue suits, with gold teeth and a
of a tyrannical conductor in a blue swallow-tailed coat, were fi
old biddy. I pull on my powder-blue sweatsuit, my disguise as a
was streaked now with midnight blue, the surrounding tissue a ra
Meissen imitations of K'ang Hsi blue-and-white: the porcelain his
ly have helped that I was a bit blue. The fact that they reserved
at topaz. Such a beautiful deep blue they 'd made it, when it mus
he used it, which was seldom. A blue threadbare carpet was spread
range, Harriet's room, with its blue-striped ticking curtains cho
ut on her eye make-up. That was blue, too. Women should never wea
e, least of all a fence, even a blue-chip top-of-the-market fence
. She blew it up for me. It was blue, translucent, round, like a
ly striped with red, yellow and blue under the turning lights. Wi
colors on a matte board: cobalt blue, violet, and rose bleeding t
burnt-orange carpet and striped blue wallpaper. However, the matt
green was more unusual, but the blue was clear and stark. The two
, Toronto the Good, Toronto the Blue, where you could n't get win
t 's most often a bolt from the blue which strikes down a good wi
s blue, and the walls were baby blue with white trim: the decor o
y call Black Irish: darkhaired, blue-eyed, with (perhaps) a strea
he steps in her ankle boots and blue woollen dress, and stood shy

APPENDIX 2: TCE PRINTOUT

Tce printout, search word *blue*. Note that the program has inserted <b> -tags around each occurrence of *blue* in order to highlight the items that were found. The program sometimes makes errors with them; in some cases, the tags have been applied to an adjacent word, not to the search word *blue*.

```
<s id=MA1.2.1.s21 corresp=MA1T.2.1.s20>First prize a week in
Toronto, second prize two weeks in Toronto, Toronto the Good,
Toronto the <b>Blue</b>, where you could n't get wine on Sundays.</s>
------------------------------
<s id=MA1T.2.1.s20 corresp=MA1.2.1.s21>Ensimmäinen palkinto
viikko Torontossa, toinen palkinto kaksi viikkoa Torontossa,
Mainiossa Torontossa, Sinisessä Torontossa, missä ei saanut
viiniä sunnuntaisin.</s>
==============================
<s id=MA1.2.3.s32 corresp=MA1T.2.3.s32>It was <b>blue</b>,
translucent,
round, like a private moon.</s>
------------------------------
<s id=MA1T.2.3.s32 corresp=MA1.2.3.s32>Se oli sininen, läpikuultava
ja pyöreä, kuin ikioma kuu.</s>
==============================
<s id=MA1.2.3.s39 corresp=MA1T.2.3.s39>The motel is the kind
we 're used to: a row of cottages, flimsily built, strung together
with Christmas tree lights, yellow or <b>blue</b> or green.</s>
------------------------------
<s id=MA1T.2.3.s39 corresp=MA1.2.3.s39>Motelli on niitä joita
me yleensäkin käytämme: rivi hataratekoisia mökkejä jotka on
yhdistetty toisiinsa joulukuusenlampuilla, <pb n=46> keltaisilla
ja punaisilla ja vihreillä.</s>
==============================
<s id=JB1.1.s127 corresp=JBF1.1.s125>A soft June morning with
a <b>blue</b> sky and a gentle breeze.</s>
------------------------------
<s id=JBF1.1.s125 corresp=JB1.1.s127>Lempeä kesäkuun  aamu,
sininen taivas ja kevyt tuulenhenki.</s>
==============================
<s id=JB1.2.s261 corresp=JBF1.2.s248>I was a bit <b>blue</b>,
and being blue always makes me satirical, so I expect the odd
unfair jest might have escaped my lips.</s>
------------------------------
<s id=JBF1.2.s248 corresp=JB1.2.s261>Olin hiukan alakuloinen,
ja alakulo tekee minut aina sarkastiseksi, joten huuliltani
on saattanut karata muutama epäoikeudenmukainen ivapuhe.</s>
==============================
<s id=JB1.2.s261 corresp=JBF1.2.s248>I was a bit blue, and being
<b>blue</b> always makes me satirical, so I expect the odd unfair
jest might have escaped my lips.</s>
------------------------------
<s id=JBF1.2.s248 corresp=JB1.2.s261>Olin hiukan alakuloinen,
ja alakulo tekee minut aina sarkastiseksi, joten huuliltani
on saattanut karata muutama epäoikeudenmukainen ivapuhe.</s>
==============================
<s id=JB1.3.s51 corresp=JBF1.3.s50>We 'd been discussing something
```

else, and I said it out of the blue, but in my mind it was as if we 'd just been talking about Oliver, and the way she answered, as if she thought we 'd just been talking about Oliver too and there was n't any break in that conversation even though we 'd been through lots of different subjects in the meantime, made me feel very cheerful.</s>

<s id=JBF1.3.s50 corresp='JB1.3.s50 JB1.3.s51'> "Käytätkö sinä meikkiä?" Olimme puhuneet jostain aivan muusta ja kysymys tuli kuin salama kirkkaalta taivaalta, mutta minusta tuntui, että olimme juuri puhuneet Oliverista, ja Gillianin tapa vastata, ikään kuin hänestäkin tuntuisi, että olimme juuri puhuneet Oliverista ja keskustelu jatkuisi siitä saumattomasti - vaikka tosiasiassa olimme käyneet läpi jo lukuisia muita aiheita - teki minut hyvin iloiseksi.</s>

==========

<s id=JB1.3.s196 corresp=JBF1.3.s191>Looking back, it might actually have helped that I was a bit blue.</s>

<s id=JBF1.3.s191 corresp=JB1.3.s196> Kun nyt muistelen sitä aikaa, luulen, että minun hienoinen alakuloni oli oikeastaan avuksi.</s>

==========

<s id=WB1.2.s141 corresp=WB1T.2.s142>Tonight he wore a faded blue polka-dot cravat at his throat which set off his tan admirably.</s>

<s id=WB1T.2.s142 corresp=WB1.2.s141>Sinä iltana hänellä oli kaulassaan haalistunut sinipilkullinen solmio, joka toi rusketuksen esiin suorastaan ihailtavasti.</s>

==========

<s id=WB1.2.3.s27 corresp=WB1T.2.3.s28> He had a long, straight nose and bright, pale blue eyes.</s>

<s id=WB1T.2.3.s28 corresp=WB1.2.3.s27> John Clearwaterilla oli pitkä suora nenä ja kirkkaat vaaleansiniset silmät.</s>

==========

<s id=ABR1.1.1.s146 corresp=ABRF1.1.1.s143>It was that blue and red, that blue especially, I felt in need of now, a need as real as that of the body when it craves fresh fruit, or vitamins, or chastity, or sex.</s>

<s id=ABRF1.1.1.s143 corresp=ABR1.1.1.s146>Juuri sitä sinistä ja punaista, etenkin sinistä, tunsin nyt tarvitsevani, tarve joka oli yhtä todellinen kuin ruumiilla kun se kaipaa tuoretta hedelmää tai vitamiinia, siveyttä tai seksiä.</s>

==========

<s id=ABR1.1.1.s146 corresp=ABRF1.1.1.s143>It was that blue and red, that blue especially, I felt in need of now, a need as real as that of the body when it craves fresh fruit, or vitamins, or chastity, or sex.</s>

<s id=ABRF1.1.1.s143 corresp=ABR1.1.1.s146>Juuri sitä sinistä ja punaista, etenkin sinistä, tunsin nyt tarvitsevani, tarve joka oli yhtä todellinen kuin ruumiilla kun se kaipaa tuoretta hedelmää tai vitamiinia, siveyttä tai seksiä.</s>

==========

<s id=ABR1.1.1.s691 corresp=ABRF1.1.1.s682>Then, quite out of the blue, it is discovered by one of the most exciting younger French directors, Francois Masson, Paul is invited to write the scenario, the film takes Cannes by storm, and suddenly

Paul finds himself established.</s>

<s id=ABRF1.1.1.s682 corresp=ABR1.1.1.s691>Sitten aivan yllättäen siihen iskee silmänsä yksi jännittävimmistä nuorista ranskalaisista ohjaajista, Francoise Masson, Paulia kehotetaan kirjoittamaan käsikirjoitus, elokuva valloittaa Cannesin yhdellä rysäyksellä, ja äkkiä Paul huomaa olevansa tunnettu mies.</s>
==========
<s id=ABR1.1.1.s751 corresp=ABRF1.1.1.s742>In Germany they later portrayed the Plague as a maid travelling through the air like a blue flame, killing her victims simply by raising an arm.</s>

<s id=ABRF1.1.1.s742 corresp=ABR1.1.1.s751>Saksassa rutto kuvattiin myöhemmin neidoksi, joka matkasi ilman halki kuin sininen liekki ja tappoi uhrinsa vain kättä kohottamalla.</s>
==========
<s id=ABR1.1.1.s936 corresp=ABRF1.1.1.s927>Spitting that Maid's blue flame right in the face and saying: "Come on, I dare you!"'</s>

<s id=ABRF1.1.1.s927 corresp=ABR1.1.1.s936>Sylkeä sen Neidon sininen liekki päin silmiä ja sanoa: 'Käy päälle jos uskallat!'"</s>
==========
<s id=BC1.6.s172 corresp=BC1T.6.s176>The four fat Party Members, at whom this outburst was directed, were far too busy to notice: they were ogling their second helping of trout whose flesh, at that moment, the waiter was easing off the bone and blue skin.</s>

<s id=BC1T.6.s176 corresp=BC1.6.s172> Neljällä lihavalla puolueen
jäsenellä, joille tämä purkaus oli suunnattu, ei ollut aikaa kuunnella; he ahmivat silmillään toista taimenannostaan, jonka lihaa tarjoilija juuri sillä hetkellä irrotteli ruodosta ja sinisestä nahasta.</s>
==========
<s id=BC1.13.s58 corresp=BC1T.13.s58>The monkeys wore ruffs and powdered wigs and, under the baton of a tyrannical conductor in a blue swallow-tailed coat, were fiddling and scraping, trumpeting, strumming and singing: in mockery of Count Brühl's private orchestra.</s>

<s id=BC1T.13.s58 corresp=BC1.13.s58> Apinoilla oli yllään röyhelöitä
ja puuteroitu peruukki, ja siniseen haarapääskytakkiin sonnustautuneen
itsevaltaisen kapellimestarin puikon johdolla ne <pb n=54> vinguttivat
ja kitkuttivat, toitottivat, näppäilivät ja lauloivat — pilkkasivat kreivi Brühlin yksityisorkesteria.</s>
==========
<s id=BC1.20.s3 corresp=BC1T.20.s3>The carpet was blue, and the walls were baby blue with white trim: the decor of the nursery, of the fresh start.</s>

<s id=BC1T.20.s3 corresp=BC1.20.s3>Matto oli sininen ja vaaleansinisissä
seinissä oli valkoiset reunukset; lastenhuoneen sisustus, uuden alun sisustus.</s>
==========

<s id=BC1.20.s3 corresp=BC1T.20.s3>The carpet was blue, and the walls were baby blue with white trim: the decor of the nursery, of the fresh start.</s>

<s id=BC1T.20.s3 corresp=BC1.20.s3>Matto oli sininen ja vaaleansinisissä
seinissä oli valkoiset reunukset; lastenhuoneen sisustus, uuden alun sisustus.</s>
==========
<s id=BC1.20.s37 corresp=BC1T.20.s37>The curtain rose on Lucienne Boyer, "La Dame en Bleu": a compact and rounded woman, approaching fifty yet apparently ageless and wearing a dress of dark blue satin, and a blue rose at the apex of her decollet,.</s>

<s id=BC1T.20.s37 corresp=BC1.20.s37> Esirippu nousi ja sen takaa ilmestyi Lucienne Boyer, <foreign lang=fr>La Dame en Bleu </foreign>; kiinteä ja pyöreä nainen, joka lähenteli viittäkymmentä, mutta näytti iättömältä, yllään tummansininen satiinipuku ja kaula-aukon pohjukassa sininen ruusu.</s>
==========
<s id=BC1.20.s37 corresp=BC1T.20.s37>The curtain rose on Lucienne Boyer, "La Dame en Bleu": a compact and rounded woman, approaching fifty yet apparently ageless and wearing a dress of dark blue satin, and a blue rose at the apex of her decollet,.</s>

<s id=BC1T.20.s37 corresp=BC1.20.s37> Esirippu nousi ja sen takaa ilmestyi Lucienne Boyer, <foreign lang=fr>La Dame en Bleu </foreign>; kiinteä ja pyöreä nainen, joka lähenteli viittäkymmentä, mutta näytti iättömältä, yllään tummansininen satiinipuku ja kaula-aukon pohjukassa sininen ruusu.</s>
==========
<s id=RD1.4.s58 corresp=RDF1.4.s60>Fred was delighted and led her up to his bedroom where a truly magnificent blue and yellow parrot sat in a tall cage.</s>

<s id=RDF1.4.s60 corresp=RD1.4.s58>Ilahtuneena Fred ohjasi hänet huoneeseensa, jossa todella upea sinikeltainen papukaija kökötti kookkaassa häkissä.</s>
==========
<s id=RD1.7.s9 corresp=RDF1.7.s9>She had a lovely pale oval madonna face with blue eyes and her hair was light-brown.</s>

<s id=RDF1.7.s9 corresp=RD1.7.s9>Hänellä oli suloiset kalpeat soikion muotoiset madonnankasvot, siniset silmät ja vaaleanruskeat hiukset.</s>
==========
<s id=RDO1.1.s1149 corresp=RDO1T.1.s1138>Stuff was growing all over them, green and blue and yellow.</s>

<s id=RDO1T.1.s1138 corresp=RDO1.1.s1149>Niissä kasvoi joka puolella jotain vihreää ja sinistä ja keltaista.</s>
==========
<s id=MD1.1.s63 corresp=MDF1.1.s66>She leaned forward, widening her pale bright blue give-away impenetrable eyes at him.</s>

<s id=MDF1.1.s66 corresp=MD1.1.s63>Kate kumartui lähemmäksi ja levitti suuriksi vaalean kirkkaansiniset vilpittömät tutkimattomat silmänsä.</s>
==========
<s id=MD1.1.s480 corresp=MDF1.1.s486>In a despair of uncertainty, she took to going round in her old navy school uniform skirt

and blue striped blouse, which passed without comment for some weeks, until Hunt said, this time not unkindly, "Going to wear those for the rest of your adult life, sweetie?"</s>

<s id=MDF1.1.s486 corresp=MD1.1.s480>Tuskaisen epävarmuuden vallassa Kate alkoi liikkua vanhassa laivastonsinisessä koulu-univormun
hameessa ja sinisessä raidallisessa puserossa, joihin ei pariin viikkoon kohdistunut huomautuksia, kunnes Hunt sanoi, ja varsin ystävällisesti sillä kertaa: "Aiotko käyttää noita koko aikuisen elämäsi, kultaseni?"</s>
==========
<s id=MD1.1.s680 corresp=MDF1.1.s685>If Evelyn felt slight misgivings as Ted and Kate greeted one another by telling one another how much they had heard about each other, with a challenging look in both pairs of bright blue eyes, she concealed them.</s>

<s id=MDF1.1.s685 corresp=MD1.1.s680> Mikäli Evelyn tunsi heikkoja epäilyksiä, kun Ted ja Kate tervehtivät toisiaan sanomalla kuulleensa toisistaan hyvin paljon, haastava ilme molemmissa kirkkaansinisissä silmäpareissa, hän salasi sen.</s>
==========
<s id=MD1.1.s683 corresp=MDF1.1.s688>Kate, for her part, was thoroughly looking forward to her evening; Stuart was baby-sitting so in theory at least she would n't have to hurry back, the spring sun was pouring through the windows onto the pale gold wooden floor and the stripped pine of the dresser and the blue bowl of orange marigolds and marguerites, it would be a good dinner because Evelyn's oven unlike her own was not bought from a junkshop and standing on a slant, the drink that Ted had poured for her was substantial, and she was delighted to find Hugo Mainwaring, one of her favourite recent acquaintances, who treated her with an asexual teasing <pb n=38>gallantry that made her feel extremely buoyant.</s>

<s id=MDF1.1.s688 corresp=MD1.1.s683>Kate puolestaan odotti iltaa innostuneesti: Stuart oli lastenvahtina joten ainakaan teoriassa hänen ei tarvitsisi kiiruhtaa heti kotiin; kevätaurinko paistoi ikkunoista vaalean kullanväriselle puulattialle, astiakaapin pelkistetylle männylle ja siniselle maljakolle jossa oli oransseja kehäkukkia ja päivänkakkaroita; ateria olisi hyvä, sillä toisin kuin heillä Evelynin liettä ei ollut ostettu romukaupasta eikä se seissyt vinossa, Tedin kaatama ryyppy oli runsas ja häntä ilahdutti tavata Hugo Mainwaring, uusi tuttava josta hän piti ja joka suhtautui häneen epäseksuaalisen ja kiusoittelevan ritarillisesti,
mikä sai hänet tuntemaan olonsa erityisen loistavaksi.</s>
==========
<s id=DF1.2.s313 corresp=DFF1.2.s311>Tears welled in her <pb n=26>uncontrollably as they had earlier, and she stared at me through them, which made her blue eyes look huge.</s>

<s id=DFF1.2.s311 corresp=DF1.2.s313>Kyynelet valuivat valtoimenaan aivan kuten aiemminkin, ja hän tuijotti minua niiden läpi, mikä sai hänen siniset <pb n=33> <blankline> silmänsä valtavan suuren näköisiksi.</s>
==========
<s id=DF1.2.s337 corresp=DFF1.2.s335>Such a beautiful deep blue
they 'd made it, when it must have been almost colourless to begin with.</s>

```
------------------------------
<s id=DFF1.2.s335 corresp=DF1.2.s337>Ne oli saatu niin kauniin
syvänsinisiksi, vaikka ne olivat alun alkaen olleet varmaan
miltei värittömiä.</s>
==============================
<s id=NG1.2.s34 corresp=NGF1.2.s34><b>Blue</b>, of course.</s>
------------------------------
<s id=NGF1.2.s34 corresp=NG1.2.s34>Siniset, tietysti.</s>
==============================
<s id=SG1.2.s108 corresp=SGF1.2.s100>Her left eye had been blackened
not long ago and it was streaked now with midnight <b>blue</b>,
the surrounding tissue a rainbow of green and <pb n=12>yellow
and gray.</s>
------------------------------
<s id=SGF1.2.s100 corresp='SG1.2.s107 SG1.2.s108'>Hänen ylähuulensa
oli turvoksissa kuin lapsella joka vasta opettelee pitämään
polkupyörää pystyssä, vasen silmä oli äskettäin isketty mustaksi
ja sitä koristi yönsinisen ja vihreän ja keltaisen ja harmaan
sekamelska.</s>
==============================
<s id=SG1.2.s150 corresp='SGF1.2.s138 SGF1.2.s139'>Lovella wore
<b>blue</b> jeans and a plain white tee shirt wrongside out,
the Fruit of the Loom label visible at the back of her neck.</s>
------------------------------
<s id=SGF1.2.s138 corresp=SG1.2.s150>Hänellä oli siniset farkut,
ja valkoinen T-paita oli väärinpäin niin että hedelmänkuvat
olivat selkäpuolella.</s>
<s id=SGF1.2.s139 corresp=SG1.2.s150>Paidan helma oli solmittu
vyötärön yläpuolelle.</s>
==============================
<s id=SG1.4.s6 corresp=SGF1.4.s6>The dawn was laid out on the
eastern skyline like watercolors on a matte board: cobalt
<b>blue</b>,
violet, and rose bleeding together in horizontal stripes.</s>
------------------------------
<s id=SGF1.4.s6 corresp=SG1.4.s6>  Aamurusko levittäytyi itäiselle
taivaalle kuin vesivärimaalaus, toisiinsa sekoittuvina
kobolttisinisinä,
violetteina ja ruusunpunaisina juovina.</s>
==============================
<s id=SG1.4.s82 corresp=SGF1.4.s80>The sunlight, intermittent
for the last hour, was now largely blocked by incoming clouds
which had crowded out <b>blue</b> sky, leaving only an occasional
patch, like a hole in a blanket.</s>
------------------------------
<s id=SGF1.4.s80 corresp=SG1.4.s82>  Viime tunnin epävakainen
auringonpaiste oli nyt peittynyt paksuneviin pilviin jotka täyttivät
lähes koko taivaan.</s>
==============================
<s id=SG1.4.s98 corresp=SGF1.4.s96>Her right eye was green,
her left eye <b>blue</b>.</s>
------------------------------
<s id=SGF1.4.s96 corresp=SG1.4.s98>Oikea silmä oli vihreä, vasen
sininen.</s>
==============================
<s id=SG1.5.s65 corresp=SGF1.5.s65>At intervals, thunder rumbled
in the distance and my windows flickered with <b>blue</b> light,
tree branches illuminated briefly before the room went black
again.</s>
------------------------------
<s id=SGF1.5.s65 corresp=SG1.5.s65>Väliin jossakin etäällä jyrähteli
```

ukkonen, ja ikkunoissa välkehtivä sininen valo sai puunlatvat kuvastumaan mustina siluetteina taivasta vasten.</s>
==========
<s id=SG1.5.s159 corresp=SGF1.5.s157>The room was small, maybe nine by nine, with four blue molded-plastic chairs hooked together at the base, a low wooden table covered with old magazines, and a television screen affixed, at an angle, up in one corner of the room.</s>

<s id=SGF1.5.s157 corresp=SG1.5.s159>Huone oli pieni, vain vajaat kolme metriä kanttiinsa, ja siellä oli neljä sinistä muovivalutuolia, vanhoilla aikakauslehdillä lastattu matala puupöytä sekä nurkassa seinään ankkuroitu televisio.</s>
==========
<s id=SG1.5.s171 corresp=SGF1.5.s168>The green was more unusual, but the blue was clear and stark.</s>

<s id=SGF1.5.s168 corresp='SG1.5.s170 SG1.5.s171'> Kaksivärinen katse nousi ja kohtasi omani ja huomasin miettiväni pidinkö enemmän sinisistä vai vihreistä silmistä.Vihreä <pb n=44> oli epätavallisempi, mutta sininen oli kirkas ja vahva.</s>
==========
<s id=SK1.1.s370 corresp=SK1T.1.s374>Her daughter turned, her daughter turned, turned, and there <pb n=16>was blood all over her mouth, it was down her chin, it was matting her blue sailor dress, blood, oh dear God dear Jesus Joseph and Mary so much <hi rend=italic>blood</hi> —</s>

<s id=SK1T.1.s374 corresp=SK1.1.s370> Hänen tyttärensä kääntyi ja verta oli joka puolella, suussa, leualla, sinisellä mekolla, verta, voi hyvä Jumala, Jeesus, Joosef ja Maria, miten paljon <hi rend=italic>verta</hi> — </s>
==========
<s id=DK1.1.2.s15 corresp=DK1T.1.2.s15> His room has burnt-orange carpet and striped blue wallpaper.</s>

<s id=DK1T.1.2.s15 corresp=DK1.1.2.s15>Hänen huoneessaan on poltetun oranssin värinen matto ja siniraidalliset seinäpaperit.</s>
==========
<s id=DK1.1.3.s2 corresp=DK1T.1.3.s2> He had Daddy's blue eyes, Daddy's dark brown hair, Daddy's too-big ears, Daddy's freckled nose; he was a dead ringer for the Martin Stillwater pictured on the dust jackets of his books.</s>

<s id=DK1T.1.3.s2 corresp=DK1.1.3.s2>Hänellä oli isän siniset silmät, isän tummanruskea tukka, isän liian isot korvat, isän kesakkoinen nenä; hän oli sen Martin Stillwaterin kaksoisolento, jonka kuva komeili kirjojen takakansissa.</s>
==========
<s id=DK1.1.3.s79 corresp=DK1T.1.3.s75> Bob was a bug, a slow-moving black beetle as large as the last joint of Daddy's thumb, with faint blue markings on his carapace.</s>

<s id=DK1T.1.3.s75 corresp=DK1.1.3.s79>Bob oli hyönteinen, isän peukalonpään suuruinen hidasliikkeinen musta kovakuoriainen, jonka selkäkilvessä oli himmeänsinisiä täpliä.</s>
==========
<s id=DK1.1.6.s2 corresp=DK1T.1.6.s2> Beneath that enormous weight of space and blackness, the Blue Life Lounge huddles like a research station on the floor of an ocean trench, pressurized to resist implosion.</s>

```
------------------------------
<s id=DK1T.1.6.s2 corresp=DK1.1.6.s2>Blue Life Lounge kyyhöttää
avaruuden ja pimeyden valtavan painon alla kuin valtameren haudan
pohjalla sijaitseva tutkimusasema, jonka sisältämä ilmanpaine
estää sitä luhistumasta kokoon.</s>
==============================
<s id=DK1.1.6.s4 corresp=DK1T.1.6.s4><b>Blue</b> and green neon
spells the name in lazy script and outlines the structure, glimmering
in the aluminum and beckoning with as much allure as the lamps
of Neptune.</s>
------------------------------
<s id=DK1T.1.6.s4 corresp=DK1.1.6.s4>Sinistä ja vihreää välkkyvä
neonvalo riipustaa nimen laiskasti virtaavana kirjoituksena
ja piirtää rakennuksen ääriviivat, heijastuu alumiinipinnasta
ja vilkuttaa houkuttelevasti kuin Neptunuksen lyhdyt.</s>
==============================
<s id=DK1.1.8.s1 corresp=DK1T.1.8.s1>In the <b>Blue</b> Life
Lounge, a woman brushes against the killer and slides on to
the bar stool beside him.</s>
------------------------------
<s id=DK1T.1.8.s1 corresp=DK1.1.8.s1>Blue Life Loungessa muuan
nainen koskettaa kevyesti tappajaa ja liukuu istumaan hänen
vieressään olevalle baarituolille.</s>
==============================
<s id=DK1.1.8.s10 corresp=DK1T.1.8.s10>  Heather tells him that
across the highway from <b>the</b> Blue Life Lounge is a motel
where, if a girl is known to the management, rooms can be rented
by the hour.</s>
------------------------------
<s id=DK1T.1.8.s10 corresp=DK1.1.8.s10>Heather kertoo hänelle,
että valtatien toisella puolella, vastapäätä Blue Life Loungea,
on motelli mistä saa vuokrata huoneita tunniksi, jos tyttö vain
on johdon tuttuja.</s>
==============================
<s id=DK1.1.8.s35 corresp=DK1T.1.8.s35>  Minutes later, after
showering and dressing, he crosses the highway from the motel
<b>to</b>-the Blue Life Lounge and gets in his rental car.</s>
------------------------------
<s id=DK1T.1.8.s35 corresp=DK1.1.8.s35>Kun tappaja on käväissyt
suihkussa ja pukeutunut, hän ylittää pari minuuttia myöhemmin
motellin ja Blue Life Loungen välisen valtatien ja nousee vuokrattuun
autoonsa.</s>
==============================
<s id=DK1.1.12.s9 corresp=DK1T.1.12.s9>He is prepared to hotwire
a car if it comes to that, but in a <b>blue</b> Honda he finds
the keys tucked behind the sun visor.</s>
------------------------------
<s id=DK1T.1.12.s9 corresp=DK1.1.12.s9>Hän on varautunut tarpeen
vaatiessa käynnistämään auton virtalukon johtojen avulla, mutta
löytää sinisen Hondan avaimet häikäisysuojuksen takaa.</s>
==============================
<s id=DK1.1.14.s38 corresp=DK1T.1.14.s38>Always wanted to see
the country, and we sure in <b>blue</b> blazes are seeing it
now.</s>
------------------------------
<s id=DK1T.1.14.s38 corresp=DK1.1.14.s38>Me ollaan aina haluttu
nähdä maata, ja nyt me totta vieköön nähdään.</s>
==============================
<s id=DK1.1.14.s56 corresp=DK1T.1.14.s54>Through the rippling
rain on the windshield, the killer watches the white-haired
man sprint to a large silver and <b>blue</b> Road King, which
```

he enters through the driver's door at the front.</s>

<s id=DK1T.1.14.s54 corresp=DK1.1.14.s56>Tappaja näkee sateen sumentaman tuulilasin läpi, miten valkotukkainen mies juoksee suuren väriltään hopeanharmaan ja sinisen Road Kingin kuljettajan puoleiselle etuovelle ja astuu sisään.</s>
===
<s id=MM1.1.s369 corresp=MM1T.1.s376>A blue threadbare carpet was spread across the floor with bits of matting added by the window and bed.</s>

<s id=MM1T.1.s376 corresp=MM1.1.s369>Lattialla oli ohueksi kulunut sininen kiinteä matto, johon oli lisätty uusia paloja ikkunan ja vuoteen eteen.</s>
===
<s id=MM1.2.s168 corresp=MM1T.2.s180>She clomped down the steps in her ankle boots and blue woollen dress, and stood shyly beside Willie, twisting the hem of her dress in her hand till her knickers came into view.</s>

<s id=MM1T.2.s180 corresp=MM1.2.s168> Tyttö tallusteli portaat alas ja jäi seisomaan ujosti Willien viereen vännellen hameenhelmaa käsissään niin että alushousut näkyivät.</s>
===
<s id=MM1.3.s39 corresp=MM1T.3.s37>"Blimmin' blue," muttered Tom to himself as he observed Willie's face.</s>

<s id=MM1T.3.s37 corresp=MM1.3.s39> "Että on sininen", mutisi Tom katsellessaan Willien naamaa.</s>
===
<s id=CP1.1.s76 corresp=CP1T.1.s78>He wore dark corduroys, a navy blue pea jacket, and a fisherman's cap, and he smelled of herring and onions.</s>

<s id=CP1T.1.s78 corresp=CP1.1.s76>Hänellä oli tummat samettihousut, tummansininen merimiestakki ja kalastajanlakki ja hän löyhkäsi silliltä ja sipulilta.</s>
===
<s id=CP1.1.s303 corresp=CP1T.1.s304>She was tall and thin and flat-chested and had pale skin and blue eyes and long fingers.</s>

<s id=CP1T.1.s304 corresp=CP1.1.s303> Hän oli pitkä ja laiha ja lattearintainen, ja hänellä oli hyvin vaalea iho ja siniset silmät ja pitkät sormet.</s>
===
<s id=CP1.1.s473 corresp=CP1T.1.s472>He was then in his middle thirties, a tall and handsome man with wavy brown hair and
blue
eyes, straight nose and strong chin, and a mouth given easily to laughter.</s>

<s id=CP1T.1.s472 corresp=CP1.1.s473>Hän oli silloin noin kolmekymmentäviisivuotias,
pitkä ja komea mies, jolla oli kiharainen ruskea <pb n=30> tukka ja siniset silmät, suora nenä ja voimakas leuka ja suu joka antautui helposti nauruun.</s>
===
<s id=CP1.1.s732 corresp=CP1T.1.s733>I sat on a bench with my mother and watched thin white clouds sail across the blue sky.</s>

```
------------------------------
<s id=CP1T.1.s733 corresp=CP1.1.s732>Minä istuin penkille äidin
kanssaja katselin ohuita valkoisia pilviä, joita purjehti sinisellä
taivaalla.</s>
==============================
<s id=CP1.1.s808 corresp=CP1T.1.s806>Unless you want to continue
straining your lovely <b>blue</b> eyes and end up bumping into
walls.</s>
------------------------------
<s id=CP1T.1.s806 corresp=CP1.1.s808>Jollet halua edelleen rasittaa
kauniita sinisiä silmiäsi ja päätyä siihen, että lopulta kävelet
päin seiniä.</s>
==============================
<s id=CP1.1.s911 corresp=CP1T.1.s906>Another boy came over,
lanky and flaxen-haired, with cold <b>blue</b> eyes and a sharp
chin.</s>
------------------------------
<s id=CP1T.1.s906 corresp=CP1.1.s911>  Paikalle tuli joku toinenkin
poika, hontelo ja vaaleatukkainen; hänellä oli kylmät siniset
silmät ja terävä leuka.</s>
==============================
<s id=CP1.1.s939 corresp=CP1T.1.s934>Cold and murderous <b>blue</b>
eyes.</s>
------------------------------
<s id=CP1T.1.s934 corresp=CP1.1.s939>Kylmät ja murhanhimoiset
siniset silmät.</s>
==============================
<s id=CP1.1.s1271 corresp=CP1T.1.s1258>It seemed to come from
the earth itself, a low enthralling tinkle of sound, like joyous
bells far off beyond the <b>blue</b> hills, beyond the green
meadows, far, far away.</s>
------------------------------
<s id=CP1T.1.s1258 corresp=CP1.1.s1271>Se tuntui tulevan suoraan
maasta, tuo lumoava hiljainen helinä joka oli kuin iloista
kellojensoittoa
kaukana sinisten vuorten toisella puolella, vihreiden niittyjen
toisella puolella, kaukana, hyvin kaukana.</s>
==============================
<s id=CP1.1.s1326 corresp=CP1T.1.s1311>Flying, flying across
the <b>blue</b> hills and crystal lakes and sunlit meadows of
my enchanted land.</s>
------------------------------
<s id=CP1T.1.s1311 corresp=CP1.1.s1326>Se lensi lentämästä päästyään
yli sinisten vuorien ja kristallinkirkkaiden järvien ja aurinkoisten
niittyjen minun lumotussa maassani.</s>
==============================
<s id=ST1.1.1.s6 corresp=STF1.1.1.s6>The Queen watched with
amused incomprehension for a while, before realising that the
red, <b>blue</b> and orange computer men represented the present
composition of the House of Commons.</s>
------------------------------
<s id=STF1.1.1.s6 corresp=ST1.1.1.s6>Kuningatar tuijotti ruutua
huvittuneen pöllämystyneenä, kunnes tajusi että punaiset, siniset
ja oranssit tietokoneukot esittivät parlamentin alahuoneen
senhetkistä
kokoonpanoa.</s>
==============================
<s id=ST1.1.9.s152 corresp=STF1.1.9.s157>He 'd seen her photograph
in the paper every day, but nothing had prepared him for the
fresh face, the soft skin, the shy <b>blue</b> eyes, the warm
damp lips.</s>
```

<s id=STF1.1.9.s157 corresp=ST1.1.9.s152>Hän oli nähnyt Dianan kuvan lehdessä joka päivä, mutta mikään ei ollut valmistanut häntä tämän raikkaille kasvoille, pehmeälle iholle, ujoille sinisille silmille ja lämpimille kosteille huulille.</s>
==========
<s id=JOT1.1.1.s125 corresp=JOT1T.1.1.s121>The stairwell was yellow and white, the sitting <pb n=9> room deep-green, the newly cxtended kitchen blue and russet and cream. </s>

<s id=JOT1T.1.1.s121 corresp=JOT1.1.1.s125>Portaikko oli keltainen ja valkoinen, olohuone syvän vihreä, vasta laajennettu keittiö sininen, punertavan ruskea ja kermanvaalea.</s>
==========
<s id=JOT1.1.2.s126 corresp=JOT1T.1.2.s125>"I mean — " She gestured at the blue frills of her nylon nightgown.</s>

<s id=JOT1T.1.2.s125 corresp=JOT1.1.2.s126>"Tarkoitan että — " Hän viittasi nailonaamutakkinsa sinisiin röyhelöihin.</s>
==========
<s id=JOT1.1.3.s75 corresp=JOT1T.1.3.s72>At Seville Airport — battered and abandoned-looking like all minor airports — a small man waited, in a blue suit.</s>

<s id=JOT1T.1.3.s72 corresp=JOT1.1.3.s75>Sevillan lentoasemalla, joka oli ränsistynyt ja hylätyn näköinen kuten kaikki pienet lentoasemat, odotti sinipukuinen, pieni mies.</s>
==========
<s id=JOT1.1.3.s132 corresp=JOT1T.1.3.s129>Would they be small and square and energetic like their driver, similarly dressed in blue suits, with gold teeth and an unshakeable view of the only kind of English person who came to Spain being that in search of sun, sangria and golf courses?</s>

<s id=JOT1T.1.3.s129 corresp=JOT1.1.3.s132>Olisivatko he pienikokoisia,
mutkattomia ja tarmokkaita kuten autonkuljettajakin, olisiko heilläkin sininen puku, kultahampaat ja vankkumaton käsitys, että kaikki Espanjaan tulevat englantilaiset hakivat sieltä aurinkoa, sangriaa ja golfkenttiä?</s>
==========
<s id=JOT1.1.3.s211 corresp=JOT1T.1.3.s209>They paced slowly by, under Frances's gaze, the dog's claws clicking on the cobbles, and then vanished round a corner where a blue neon sign said, "Bar El Nido" with a helpful indicating arrow.</s>

<s id=JOT1T.1.3.s209 corresp=JOT1.1.3.s211>He astelivat hitaasti ohi Francesin katsellessa — koiran kynnet rapsahtelivat mukulakivillä — ja katosivat sitten kulmauksesta, missä sininen neontaulu julisti suuntanuolen avustamana: Bar El Nido.</s>
==========